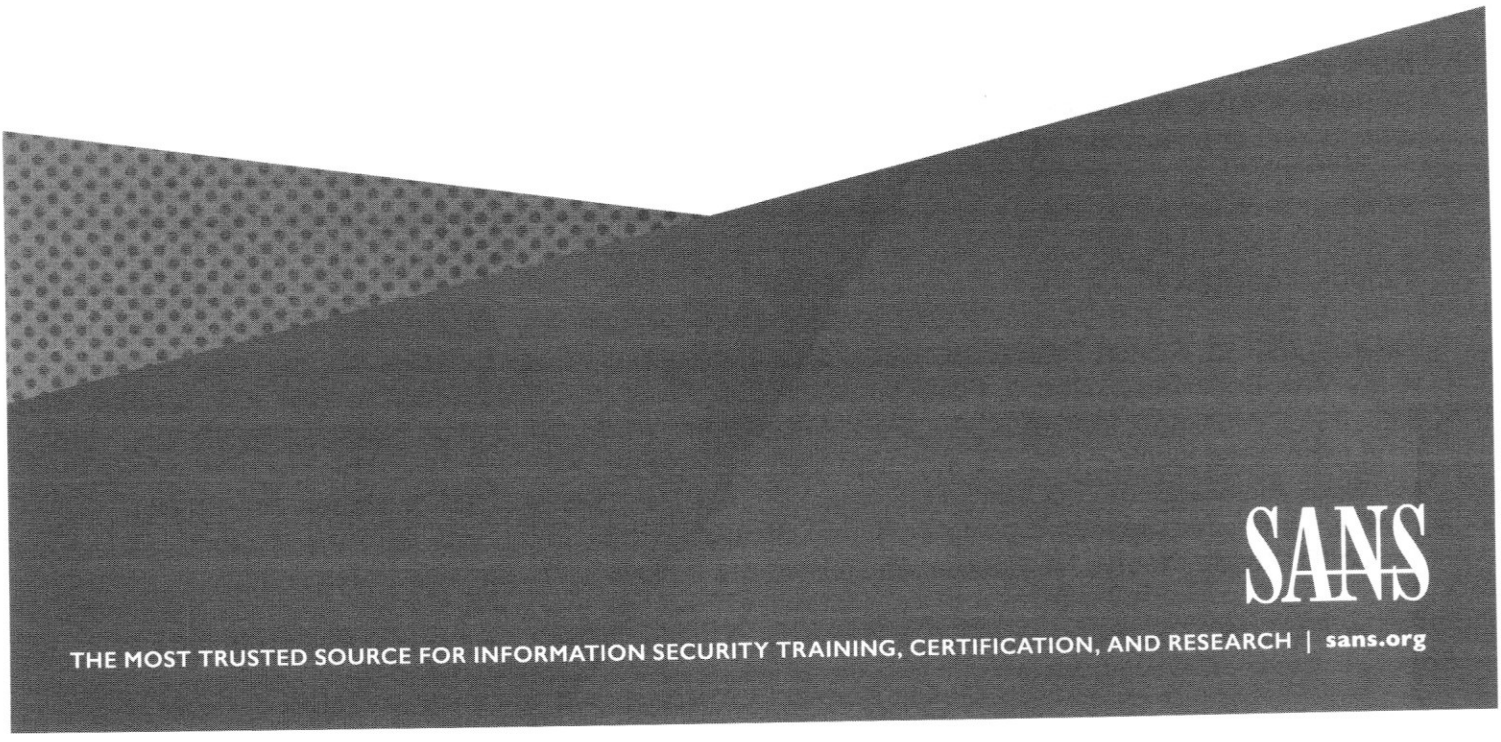
540.2
Moving to Production



SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

DEV540.2 | Secure DevOps & Cloud Application Security

SANS

Moving to Production

This page intentionally left blank.

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- *Secure Infrastructure as Code*
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.

What Is Infrastructure as Code?

Different approaches to set up and manage systems

1. Traditional: manual checklists and scripts, ad hoc changes/fixes made by system administrators at runtime
2. Modern: treating Infrastructure as Code and configuration management as system engineering

Configuration management with scripts is not scalable or traceable and is error prone... which leads to configuration drift over time

Configuration in code can be standardized, scaled, and audited

- Configuration is checked in and version controlled, reviewed and tested
- Changes and patches rolled out through the same automated pipelines
- Standardized across and within environments: **cattle, not pets**

Treating Infrastructure as Code is one of the most fundamental and important changes in DevOps. Instead of relying on system administrators to set up systems through scripts and checklists, DevOps teams write configuration definitions and policies directly into code using tools like Ansible, Chef, and Puppet.

This configuration code is checked in to repos, reviewed, refactored, scanned, and tested like any other code. Changes are pushed out through the same kind of Continuous Delivery pipelines as application changes. This provides the same controls over quality, transparency, repeatability, and traceability for all changes. Security patches and package upgrades are also pushed out using the same approach.

Server configuration is standardized between and within environments. Server instances are treated like cattle, commodities that exist only for a short time, instead of pets that are carefully pampered for years. Operations standards and compliance and security policies can be codified and implemented the same way.

From Infrastructure to Code

There are significant challenges and obstacles to getting your existing infrastructure into code:

- Legacy systems: no/outdated documentation on policies, limited platform support
- Snowflakes: one-off configurations and inconsistencies that need to be identified, understood and resolved
- Rewriting all of the directives into code
- Testing to make sure that you got it all right
- Establishing a software engineering culture for operations: version control, coding standards, reviews, refactoring, testing...
- Automating all of the steps into CI/CD

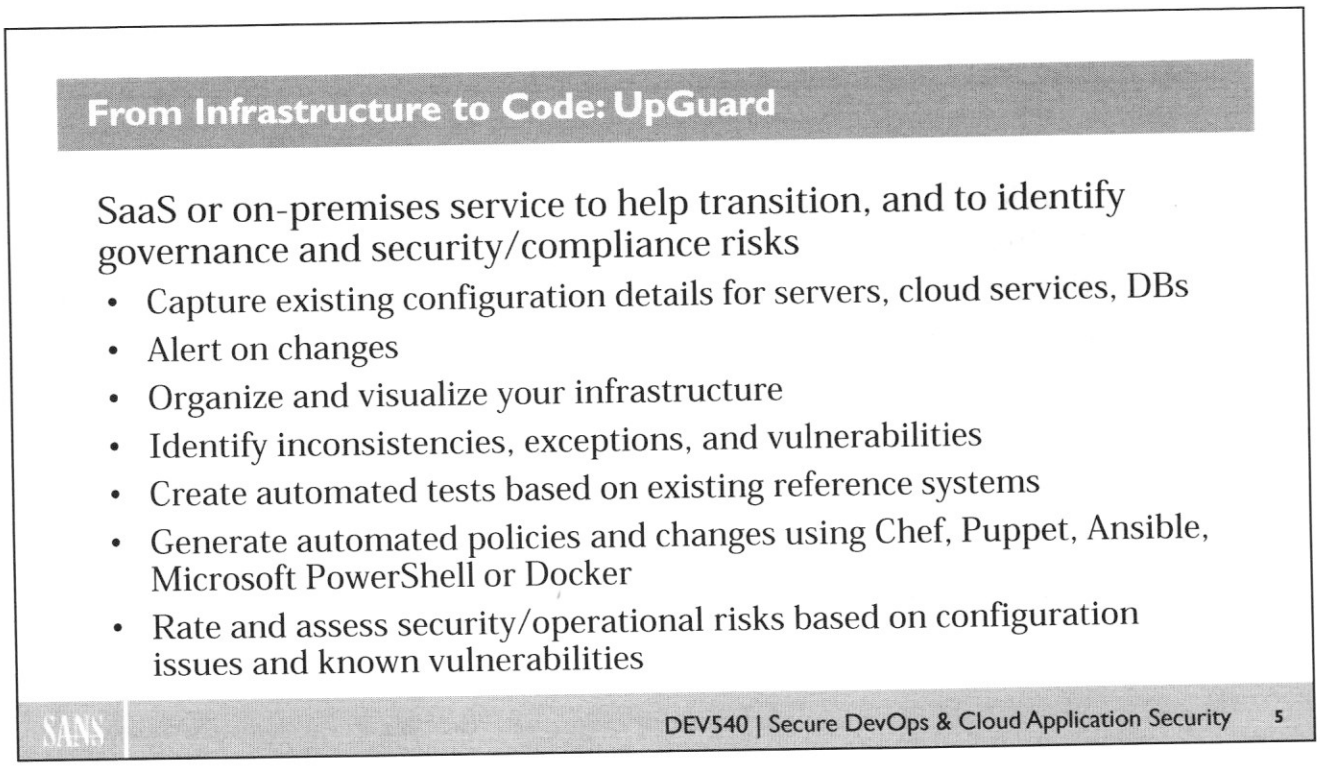
Infrastructure as Code is a new way of working for many system administrators. It is less about system administration, and more about software engineering:

- Establishing and following coding standards and patterns, and defining and enforcing security and compliance policies in code
- Organizing code into modules, and creating reusable functions and templates
- Implementing static analysis/linting tools to catch common mistakes and enforce good coding practices/style
- Checking all code/configuration into version control
- Establishing team disciplines of code reviews and refactoring
- Writing automated tests to prove that the changes are implemented correctly - including Test Driven Development (TDD) where tests are written before the code is changed (red/green)

Moving from Infrastructure to Code takes a lot of work:

- Identifying all of the different systems
- Understanding and capturing all of the configuration details
- Finding and dealing with inconsistencies
- Implementing a configuration management tool
- Rewriting the configuration specifications into code
- Writing tests to prove that the code is working
- Creating a CD pipeline/workflows to handle and implement changes

It can take months or even years to get all of this done in large enterprise/legacy environments.



A startup called UpGuard (fka ScriptRock) can help with moving from Infrastructure to Code (https://www.upguard.com/)

UpGuard is a service (on-premises or Cloud-based) that will capture system configuration for physical or virtual servers, Cloud services, and databases. You can visualize this information, compare it over time and alert on changes, group systems together and compare them to find inconsistencies in configuration and to find vulnerabilities.

You can also establish policies for different systems or types of systems (defining what settings should be on or off, what packages should be installed and what versions, what files should exist or not exist, what ports should be opened or closed, etc.) and create automated tests to check these policies. UpGuard can generate instructions to implement configuration policies, which can be exported into Chef, Puppet, Ansible, Microsoft PowerShell DSC or Docker.

UpGuard assesses the configuration of servers, network devices, and cloud apps and calculates the potential risk for intrusions and outages, assigning a "FICO-like score called CSTAR – Cyber Security Threat Assessment Report" (https://www.upguard.com/cstar).

Facebook's OSQuery Tool

Programmatic SQL interface to ask questions about Linux / OSX and Windows systems and return results in format that other tools can use

- Detective change control/intrusion detection
- Compliance auditing and governance insight
- Forensic analysis and investigation

Provides a consistent interface to different configuration data

Can join different kinds of data into coherent views

Ad hoc or scheduled

Extensible through plugins

OSQuery (https://osquery.io/) is an open source tool to ask questions about Linux/OSX and Windows systems using a SQL interface, returned in table form. You can create complex queries which join together different kinds of system information such as process status, loaded kernel modules, installed software, active network connections, user information, password changes, firewall exceptions…

Queries can be run on an ad hoc basis or scheduled automatically through a daemon (osqueryd). This allows you to automate system checks and provides a powerful tool for forensic analysis and investigation.

https://code.facebook.com/posts/844436395567983/introducing-osquery/

http://www.eweek.com/security/facebook-enhances-osquery-security-analysis-platform.html

http://www.businessinsider.com/facebook-security-open-source-osquery-cyber-attacks-2015-10

http://www.wired.com/2014/10/facebook-builder-osquery/

https://puppet.com/blog/sysadvent-so-server-tell-me-about-yourself-%E2%80%94-an-intro-to-facter-osquery-sysdig

https://osquery.io/docs/tables/

Host Intrusion detection with OSQuery https://speakerdeck.com/marpaia/host-intrusion-detection-with-osquery

Yahoo's security team, for example, has audited OSQuery https://yahoo-security.tumblr.com/post/118445868880/osquery-security-audit#_=_

Infrastructure as Code – Tooling

Tool	Configuration Approach
Chef	Mostly imperative
Puppet	Mostly declarative
Ansible	Hybrid
Salt/Saltstack	Hybrid
CFEngine	Declarative

Declarative vs. Imperative configuration approaches: define the end state vs. the detailed steps and logic required to make a change

Modern "Infrastructure as Code" configuration management tools share some common features and capabilities:

- Code-driven, using recipes or manifests to programmatically define configuration steps (imperative) or configuration state (declarative)
- Code and templates for common configurations are available in public repositories for downloading
- Provide centralized management and reporting of changes across a network
- Auditing and logging of all actions: what was changed, who made the change, when
- Offered in Open Source and commercial Enterprise versions (including extra administrative and reporting features, higher scalability, and support)
- Work on Linux, some Unix, Mac OS X (usually) and Windows platforms, as well as some cloud platforms

For a comprehensive list of all open source configuration management tools, see:
https://en.wikipedia.org/wiki/Comparison_of_open-source_configuration_management_software

Declarative vs Imperative models

Configuration details are defined using 2 different methods:

1. **Declarative**: you describe the end state that you want, not the steps to achieve it. What, not How.
2. **Imperative**: you describe the steps to set up the configuration that you want. How, not What.

Puppet is mostly declarative. Chef is mostly imperative. Ansible and Salt are hybrids.

Infrastructure as Code - Chef

One of the most popular configuration management tools

- Flexible, mostly imperative tool – define the steps to accomplish your end goal
- Ruby DSL – configuration recipes organized into cookbooks
- Large community of users
- Large ecosystem of tools and add-ons
- Enterprise and free versions available
- Community-built templates in Chef Supermarket
- Strong test tool support

Popular with developers

Chef (https://www.chef.io/chef/)

A leading tool (customers include Facebook, GE, and Disney), more oriented towards developers. Configuration is defined programmatically in Ruby with a simple DSL ("recipes" organized into "cookbooks'). Chef offers more programmatic flexibility than Puppet.

Both Chef and Puppet have a large community of users, and there are lots of pre-built recipes/manifests available from the vendors and the community: see the Chef Supermarket (https://supermarket.chef.io/) or the Puppet Forge (https://forge.puppet.com/) repositories.

Infrastructure as Code - Puppet

One of the most popular configuration management tools

- Large community of active users
- Strong tooling support
- Community-built templates in Puppet Forge
- Declarative language built on Ruby, designed to be familiar to system administrators
- Master/Agent model
- Open source and commercial enterprise versions available

In this course, we will be doing labs using Puppet – deploying a package, making a simple configuration change

Puppet (https://puppet.com/product/how-puppet-works)

One of the most popular configuration management tools: used by Salesforce, Sony, and Google. Configuration is defined declaratively using a custom Ruby DSL (Domain-Specific Language) in "manifests" organized into modules. Puppet's declarative language is designed to be familiar to system administrators and was originally based on the Nagios configuration file format.

Configuration changes are made from one or more centralized "Puppet Master" systems to remote agents.

Both Chef and Puppet have a large community of users, and there are lots of pre-built recipes/manifests available from the vendors and the community: see the Chef Supermarket (https://supermarket.chef.io/) or the Puppet Forge (https://forge.puppet.com/) repositories.

Podcasts on Puppet and Security

https://puppet.com/podcasts/puppet-podcast-security-luke-kanies-and-chris-barker

https://puppet.com/podcasts/security-and-devops-puppet-podcast

Infrastructure as Code – Ansible

Easy to set up and easy to use configuration management tool

- Executes arbitrary commands over SSH
- Does not need agents installed on each machine
  ...but does need Python
- Can be used to execute ad hoc SSH commands across a network
- Configuration is done in Playbooks using YAML
- Does not need a master: can be run from any admin system

Enterprise version is called Ansible Tower (from Red Hat)

Ansible (http://www.ansible.com/)

Easy-to-use configuration management tool, originally developed in Red Hat's emerging technology group and acquired by Red Hat in late 2015. Ansible is written in Python, and works as a thin wrapper to execute commands over SSH (or native PowerShell remoting on Windows): any node that has the correct list of servers can be used to push out changes (copy files, install or remove packages, execute commands) over SSH to the rest of the network. Agents do not need to be setup – all that is required is Python.

Configuration is defined using in "Playbooks" using YAML http://docs.ansible.com/ansible/playbooks.html)

Best practices guide: http://docs.ansible.com/ansible/playbooks_best_practices.html

https://www.ansible.com/blog/ansible-best-practices-essentials

Commands can be tested in "Dry Run" or "Check Mode" – which will trace execution but not run the commands

http://docs.ansible.com/ansible/playbooks_checkmode.html

"Ansible: Up & Running", Lorin Hochstein http://shop.oreilly.com/product/0636920035626.do

"Ansible for DevOps", Jeff Geerling https://leanpub.com/ansible-for-devops

Infrastructure as Code – Salt/Saltstack

Salt from Saltstack is the newest configuration manager on the block – popular with the cool kids

- Uses fast messaging (ZeroMQ) to collect configuration data "grains" or execute commands on remote systems – called minions
- Like Ansible, supports ad hoc command execution
- Developed in Python, configuration is defined in YAML "Salt States"
- Available in community (Salt Open) and Enterprise versions
- Highly scalable – used at LinkedIn for example

Check out **Hubblestack**, an open source configuration auditing tool built on SaltSatck

Salt (http://saltstack.com/)

Salt is built on a fast, efficient and scalable parallel remote command execution framework – instructions are executed over encrypted ZeroMQ connections between one or more masters and the remote systems to be managed (called "minions"). Salt can be used to execute ad hoc commands across many different remote systems, gather information (called "grains") or apply configuration changes to the remote systems, spin up or shut down cloud instances, or proactively monitor the network.

Salt is written in Python. Configuration ("Salt States") is defined in YAML using Jinja2 templating. Sensitive information (database passwords, keys etc) are stored in "pillars" so that they can be shared with minions.

Because it leverages ZeroMQ, Salt is highly scalable. Reference installations include LinkedIn.

Guidelines for securing your Salt configuration: https://docs.saltstack.com/en/latest/topics/hardening.html

Salt Open community: https://saltstack.com/community/

Hubblestack (https://hubblestack.io/) is an open source auditing framework built on SaltStack. Hubblestack provides on-demand profile-based auditing, real-time security event notifications, automated remediation, alerting and reporting.

Infrastructure as Code – CFEngine

The first programmable configuration manager – goes back to 1993

- Still used by big banks and enterprises
- Configuration is done through "promises"
- Written in C, highly efficient, but harder to learn and use than more modern tools
- Open source and enterprise versions
- Community and toolchain support is not as strong as other tools

Can be used to detect file changes (like Tripwire) and to enforce configuration policies

CFEngine (http://cfengine.com/)

CFEngine is the oldest programmable configuration management tool (originally developed back in 1993). It is used by a number of big banks and online enterprises including LinkedIn. CFEngine is written in C and has the smallest runtime footprint of all of the tools. Configuration is defined through "promises".

Best practices:

http://watson-wilson.ca/blog/2014/07/04/cfengine-best-practices-testing/

http://watson-wilson.ca/blog/2014/07/18/cfengine-best-practices-part-2/

http://watson-wilson.ca/blog/2014/08/07/cfengine-best-practices-deployment-upgrades-and-scaling/

Security scanning and Tripwires with CFEngine: https://docs.cfengine.com/docs/archive.bak/st-scan.html

http://www.linuxjournal.com/article/10924

Presentation and resources on applying security hardening using CFEngine:
https://www.slideshare.net/MichaelClelland/security-practices-with-cfengine-config-management-camp-2016

How to achieve DISA STIG compliance with CFEngine 3: https://docs.cfengine.com/docs/archive/stig.html

Watch: "Strategies for improving security of your IT infrastructure with CFEngine"
https://www.youtube.com/watch?v=7gb7ic0T708

"Learning CFEngine 3: Automated system administration for sites of any size", Diego Zamborni
http://shop.oreilly.com/product/0636920022022.do

Provisioning Development and Test Environments: Vagrant

Rapid and simple configuration and provisioning of VMs for development/testing

Use cases:

- Create developer environments aligned with production
- Spin up temporary disposable VM sandbox for testing Chef/Puppet changes
- More useful for a small number of VMs

Integrates with configuration management software (Chef, Puppet, Ansible, Salt...) and with Hashicorp's Packer (next slide)

Vagrant (https://www.vagrantup.com/) is an open source tool for rapidly configuring and spinning up VMs. It is a common part of Agile and DevOps automation toolsets. Vagrant can support a range of different runtime environments: VirtualBox, VMWare, KVM, Linux Containers (LXC), cloud platforms including EC2, and Docker.

A Vagrant environment is defined using a simple declarative scripting model. It also integrates with configuration management tools like Chef or Puppet to install packages and build standard configurations.

Some common use cases for Vagrant include:

- Creating self-service virtualized developer environments that align (as closely as possible) with production
- Spinning up temporary/ephemeral virtualized test environments for testing, including testing Chef/Puppet/Ansible… changes that could be destructive

Read: "Stronger DevOps Culture with Puppet and Vagrant" by Mitchell Hashimoto (principal author of Vagrant) https://puppet.com/blog/stronger-devops-culture-puppet-and-vagrant

"DevOps Technologies: Vagrant" SEI Series on DevOps https://insights.sei.cmu.edu/devops/2014/12/devops-technologies-vagrant.html

Provisioning Development and Test Environments: Packer

- Packer: open source tool for creating machine and container images on different platforms from a common configuration
- Build images for Docker, VMWare, Virtual Box
- ... and images for cloud platforms including Amazon EC2, Google Compute Engine, Microsoft Azure
- Integrates with Chef, Puppet, etc.

Packer (packer.io) from Hashicorp (inventors of Vagrant) is a packaging tool that can be used to bake machine images on multiple different platforms. You can use it to create EC2 AMIs, VMware VMDK/VMX files, Docker images, etc.

You can use the same image to deploy in Docker or VMWare and Vagrant on a local system for development, staging/QA in a private OpenStack cloud, and production in AWS or Google Compute Engine.

Infrastructure as Code – Testing

Automated tests and syntax checking are critically important for configuration code because this code is written in scripting languages with dynamic typing/content – which means problems are hard to find before runtime

Different kinds of testing tools:

- Syntax checking/linting to find typing mistakes and bad practices
- Unit testing: mock or stub out the runtime environment and check that the logic is consistent (fast checks)
- Acceptance testing – model the expected final state of the configuration, run the changes in a test environment, and compare the actual state (slow, expensive checks – but the most valuable)

Automated tests are important in catching mistakes in configuration, and catching regressions as you make changes or refactor code. This is especially important with configuration code because this code is written in scripting languages with dynamic content and dynamic typing: you won't be able to find mistakes in the code without careful inspection or running it to see what breaks. Just like application code, testing should be done at different levels:

Syntax Checking / Linting

- Puppet: puppet parser validate, puppet-lint
- Chef: foodcritic, rubocop
- Ansible: ansible-lint

Unit Testing

- rspec-puppet
- chefspec

Acceptance Testing

- serverspec (http://serverspec.org/) – tool-agnostic declarative DSL
- RSpec (https://github.com/rspec/rspec) - roll your own test framework, instead of using serverspec extensions
- beaker for Puppet (https://github.com/puppetlabs/beaker)
- Goss (https://github.com/aelsabbahy/goss) YAML-based test framework written in Go
- InSpec (https://www.inspec.io/) – compliance extension to serverspec
- Bats (https://github.com/sstephenson/bats) – Bash Automated Testing System

Vagrant and Docker can be used to quickly spin up virtual machines for doing integration and acceptance testing of recipes or cookbooks.

A good overview of tools for testing Chef and Puppet: http://ec2dream.blogspot.com/2014/02/test-driven-development-with-chef-and.html

Testing Chef Cookbooks: http://technology.customink.com/blog/2012/08/03/testing-chef-cookbooks/

Testing Puppet Manifests: https://puppet.com/blog/verifying-puppet-checking-syntax-and-writing-automated-tests

Testing Ansible Playbooks: http://docs.ansible.com/ansible/test_strategies.html

Infrastructure as Code – Test Driven Infrastructure (TDI)

Test-Driven Infrastructure – using TDD (Test-Driven Design/Development) principles to drive infrastructure definition

- Write acceptance tests first – starting by modeling the outcome
- Execute tests – will fail
- Write the code
- Execute tests – will pass

The tests act as a specification, to guide your thinking as you make changes, and as documentation for the infrastructure – as well as regression safety harness for future changes

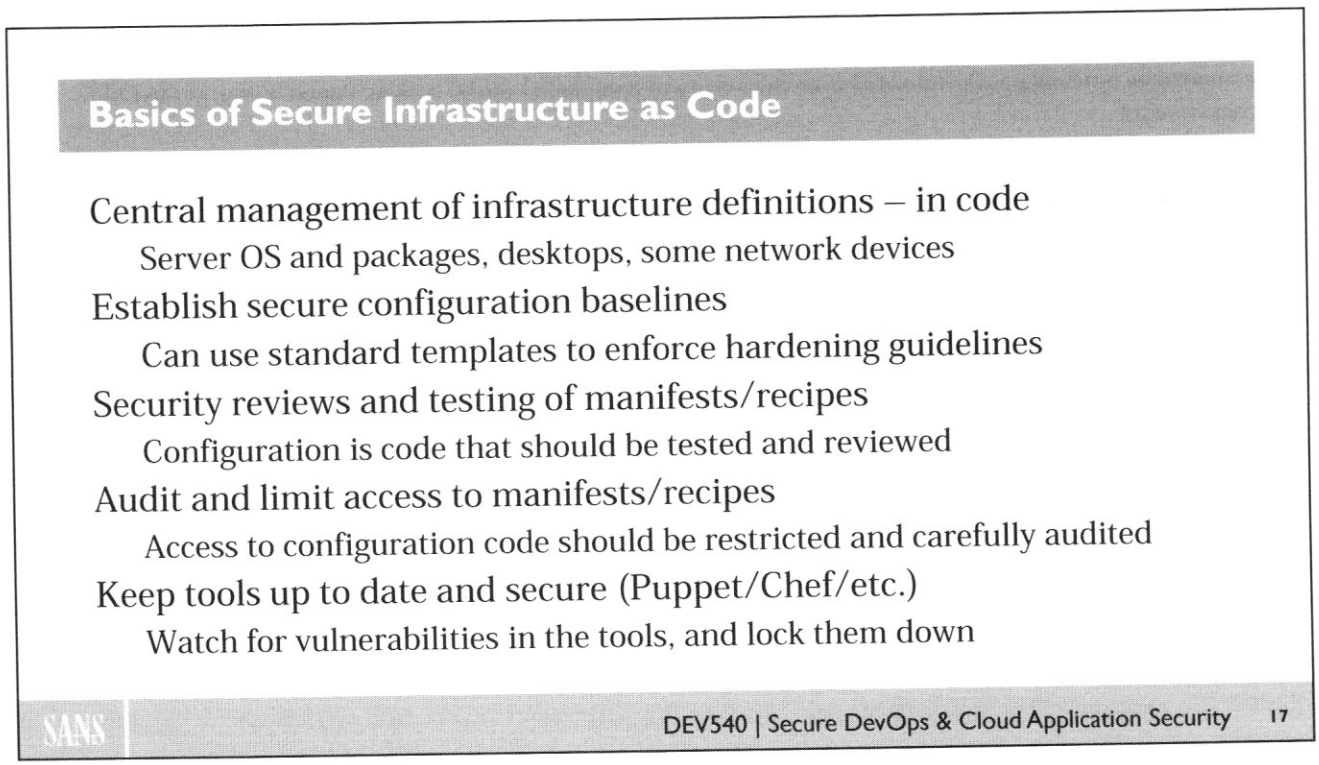
Test-Driven Infrastructure

Test-Driven Infrastructure is based on the same ideas and practices as Test-Driven Design/Development:

1. Write the tests first, as a specification of what you want the configuration outcome to be. This is best done using an acceptance testing framework like serverspec.
2. Execute the test, see that it fails – to make sure that you wrote the test correctly.
3. Write the code.
4. Execute the test, and make sure that it passes.

Test-driven approaches to development ensure a high level of test coverage. The tests serve as a specification, as well as a way of documenting the infrastructure configuration.

"Test Driven Development with Puppet" https://puppet.com/presentations/test-driven-development-puppet-gareth-rushgrove-puppet-labs
"Test-Driven Infrastructure with Chef" http://shop.oreilly.com/product/0636920030973.do



Tools like Ansible, Chef, and Puppet allow you to centrally define and provision system configurations for Linux and Windows servers and desktops, and for some network devices. Configurations are written as code, and then applied automatically to systems under management.

This provides some obvious security advantages:

- System configuration is maintained in code, which can be reviewed and tested before the changes are applied, and maintained in version control
- You can leverage the same Continuous Integration/Continuous Delivery pipeline and automated testing infrastructure for configuration changes as you do for application code, pushing changes through test
- System configuration is fully transparent - you know exactly how each system is defined, and you can compare the actual runtime definitions against the configuration database and report on variances
- You can ensure that systems are set up consistently – the chances of a system being missed or misconfigured are low, preventing snowflakes and configuration drift
- You can build hardening guidelines directly into configuration instead of executing separate scripts or following manual checklists, and audit configurations against hardening guidelines (like CIS)
- An audit trail is available for every configuration change
- It is easy to identify systems which need to be patched or upgraded
- Changes can be pushed out quickly, even immediately or automatically in many cases – for example, tools like Puppet check the configuration of systems every 30 minutes by default, and can automatically detect that a system should be patched and apply the patch immediately
- Changes can also be rolled-back quickly in the event of a problem

InfoSec needs to be involved in establishing secure configuration baselines, and in reviewing changes to configurations as they are made, and before they are checked-in. InfoSec should also review and approve any components or templates downloaded from third party or community repos, such as Puppet Forge or the Chef Supermarket.

Access to manifests/recipes in source control should be audited, and limited to people who have a need to work with the code.

It is also important to ensure that the configuration management tool chain is set up securely and to keep the tools up to date with the latest patches.

Infrastructure as Code – Hardening

Follow basic good practices in setting up configurations

...and take advantage of hardening templates/recipes and examples

- CIS benchmarks
- STIG templates using Puppet
- Community guidelines
- Hardening framework from Deutsche Telekom and others for Ansible, Chef, and Puppet (http://dev-sec.io)
- SIMP Project from NSA (https://simp-project.com/)

Always carefully review open source templates/modules before use

Basic good practices should always be followed in configuring systems, whether you are using scripts or manual checklists, or a configuration management tool such as Ansible, Chef or Puppet. There are a number of different guidelines, and pre-prepared cookbooks/manifests for hardening systems, including Center for Internet Security (CIS) modules that can be used as references:

Puppet module to implement CIS for RHEL 6/7:

https://github.com/arildjensen/cis-puppet

https://www.safaribooksonline.com/library/view/learning-puppet-security/9781784397753/ch06s03.html

http://grokbase.com/t/gg/puppet-users/15296tt97y/any-pointers-to-rhel7-cis-hardening-usig-puppet

SIMP hardening system in Puppet: https://simp-project.com/

CIS Benchmark Remediation Kits (for CIS members only) include Puppet modules for RHEL and Centos

https://benchmarks.cisecurity.org/downloads/remediation-content/

Puppet and NIST 800-53

Los Alamos case study Linux and Mac OS X hardened configurations to meet NIST guidelines

http://www.slideshare.net/PuppetLabs/los-alamos-case-study-puppet-labs

USGCB – US Government Configuration Baseline
NIST standard definition for RHEL Linux desktop configuration, defined in Puppet modules
http://usgcb.nist.gov/usgcb/rhel_content.html

Dev-Sec.io: hardening framework from Deutsche Telekom for Chef, Puppet and Ansible (http://dev-sec.io/)
https://github.com/dev-sec/hardening

Hardening with Ansible
https://github.com/ansible/ansible-lockdown
http://www.linuxjournal.com/content/security-hardening-ansible
https://www.ansible.com/security-stig
https://www.ansible.com/blog/stig-automation
https://github.com/major/cis-rhel-ansible

Hardening with CFEngine
http://security-ingvar-ua.blogspot.ca/2014/04/using-cfengine-for-linux-systems.html
http://cfengine.com/solutions/industrial-iot-systems-hardening/

DoD Security Technical Implementation Guide (STIG) templates
http://iase.disa.mil/stigs/Pages/index.aspx
https://github.com/fcaviggia/hardening-script-el6
https://github.com/robertmaury/stig
http://unix.stackexchange.com/questions/138086/stig-security-technical-implementation-guides-automation

How to establish secure infrastructure baselines in Puppet/Chef
http://www.youtube.com/watch?v=wKp32yWSkns
https://puppet.com/presentations/auditingsecurity-puppet-robert-maury-puppet-labs

Network as Code

You can use the same patterns, techniques (and many of the same tools) to configure and manage network devices

Ansible (especially), Chef, Puppet and Salt support a range of network devices with prebuilt modules (usually from vendors)

- Provision, update and configure network firewalls, load balancers, routers and switches in code
- Common language between ops and network engineering (and across heterogenous devices)
- Coordinate changes to servers + storage + network
- Review and test changes in advance, and apply changes at scale
- Continuously monitor and enforce policies from central point

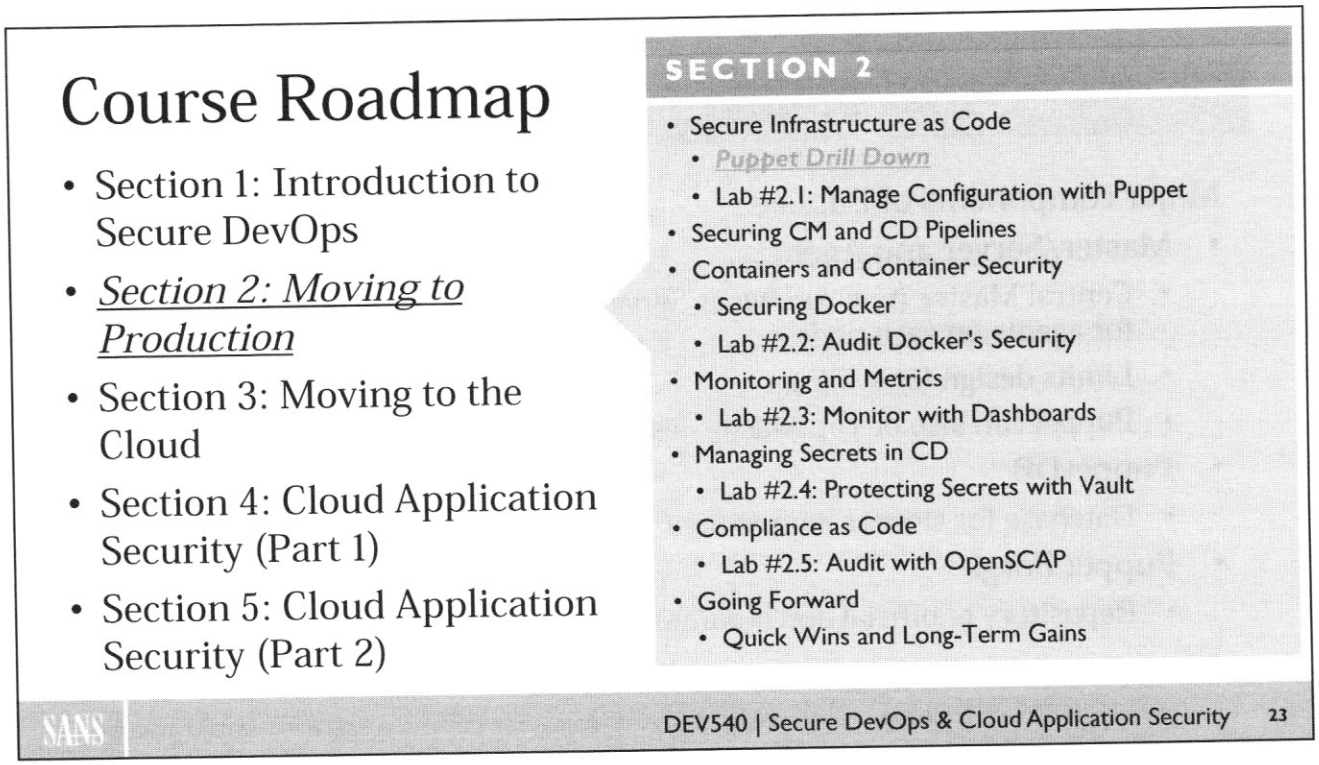
Configuration management tools like Ansible (especially), Chefm Puppet and Salt can also be used to manage network devices (firewalls, load balancers, switches and routers) in the same way as server infrastructure and cloud infrastructure. This provides all of the same benefits:
-Configuration in code – which can be version controlled, reviewed, and reused
-Central management and validation of configuration across network devices
-Apply patches and hardening steps efficiently and consistently, across many devices
-Automated testing of changes – and audit trail of when changes were applied
-Continuous monitoring and enforcement of policies to prevent configuration drift
-Makes it easier to coordinate changes to servers, storage and network devices (and applications…)

Ansible:
Ansible's agentless, SSH command execution framework makes it a natural fit for managing network devices. Strong support for network management across most major platforms: Arista, Avi Network, Cisco, Cumulus, Dell EMC, F5, Juniper, Nokia, Lenovo, and Pluribus https://www.ansible.com/network-automation

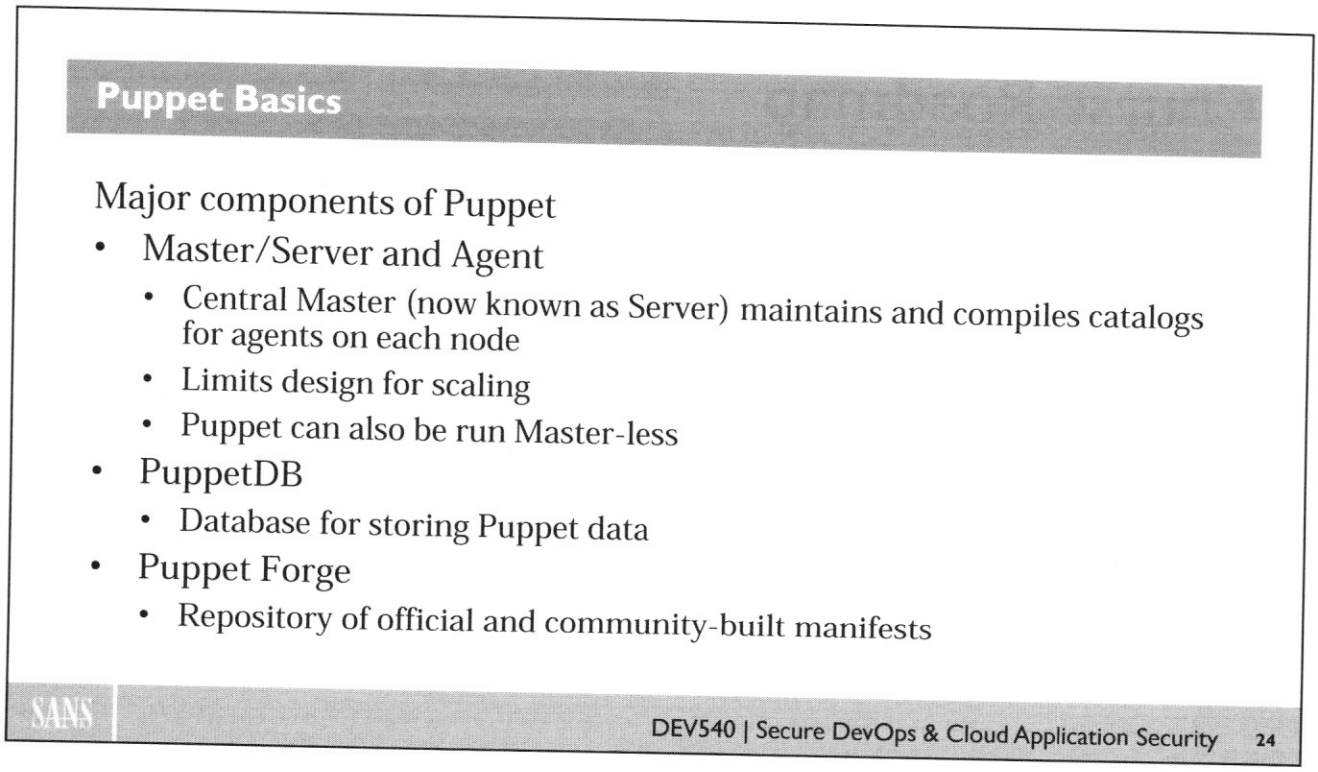
Salt:
Proxy minions can be used to control network devices vis SSH, HTTP. Salt supports Juniper JunOS, Cisco NXOS, Cisco NSO, and NAPALM
https://www.reddit.com/r/networking/comments/53djh9/salt_for_network_config_automation/
https://docs.saltstack.com/en/latest/ref/proxy/all/salt.proxy.nxos.html
https://docs.saltstack.com/en/latest/ref/proxy/all/salt.proxy.cisconso.html
https://docs.saltstack.com/en/latest/ref/modules/all/salt.modules.junos.html
https://docs.saltstack.com/en/latest/ref/proxy/all/salt.proxy.napalm.html

Chef:
Arista, Cisco and Juniper devices
https://github.com/aristanetworks/chef-eos
https://github.com/cisco/cisco-network-chef-cookbook
https://docs.chef.io/junos.html
https://github.com/chef-partners/netdev

Puppet:
Arista, Cisco, F5, Juniper
https://puppet.com/solutions/networking-automation
https://puppet.com/blog/network-automation-configuration-management-isn%E2%80%99t-just-for-compute-servers
https://puppet.com/product/managed-technology/arista
https://puppet.com/product/managed-technology/cisco
https://puppet.com/product/managed-technology/f5
https://www.juniper.net/documentation/en_US/release-independent/junos-puppet/information-products/pathway-pages/index.html



This page intentionally left blank.



Puppet can effectively run on any platform that supports a Ruby runtime: RHEL and Centos, Debian, Ubuntu, Fedora, other 'nixs, Mac OS X, Windows

Master-Agent model

1. Agents run on each node being managed with Puppet. By default, Agents check in with the Master every 30 minutes. Agents wake up, capture, and report state information using a utility called facter. This includes information about the runtime OS and hardware, available memory, IP addresses and other network information, and information about the Ruby runtime.
2. The Agents contact the Master and pass the state information to the Master.
3. The Master will use this state information to build a catalog for the Agent that describes the desired configuration for that node and sends this catalog to the Agent.
4. The Agent compares the desired state described in the catalog to the actual runtime system configuration. If there are differences, the Agent will update the runtime configuration to match the catalog.
5. All of the results are reported back to the Master.

Agents can also run in reporting mode using --noop

The Agent will compare the actual configuration to the catalog definition, and report any variances – useful for detective change control and auditing purposes.

The Puppet Master (or Puppet Server) is a central manager that maintains catalogs and communicates with Agents. It maintains configuration information for every node and compiles catalogs for each node.

The Master records the results of each Agent run and can forward this to a separate database service such as PuppetDB (https://docs.puppet.com/puppetdb/). PuppetDB also maintains information about the state of each node, making reporting and auditing easy. To improve scalability, the Puppet Master has been replaced by a JVM-based multi-threaded Puppet Server (https://github.com/puppetlabs/puppetserver). Puppet Enterprise 3.7 and later use Puppet Server by default.

Masterless Puppet

Agents can also be run in a masterless configuration. This is useful for:

- Test and development – where the Puppet Agent is invoked manually
- At the other extreme, where the runtime environment needs to scale to handle a large number of nodes – beyond the capability of the central Puppet Master/Server to handle. In this case, you need to separately command the Agents to check out new catalogs and run them, using MCollective or some other mechanism.

MCollective – the Marionette Collective, is command and control software that can be used to control multiple nodes. It supports different command plug-ins, including one for Puppet. MCollective can be used to execute arbitrary commands across managed servers – this needs to be carefully secured.

Puppet Forge

Puppet Forge (https://forge.puppet.com/) is an open source community repository of Puppet code that you can download and use, or use as examples. You can find Puppet modules contributed by Puppet Labs at https://forge.puppet.com/puppetlabs

Caveat emptor: it's up to you to review and make sure that any code that you download is up-to-date and safe-to-use

Puppet Enterprise

Puppet also offers a commercially-supported, enterprise edition of Puppet. It includes additional configuration management, node management, and code management dashboards and reporting features and role-based access control to make it easier to manage a large number of servers. See https://puppet.com/product/puppet-enterprise-and-open-source-puppet

Puppet Basics: The Puppet DSL

The Puppet Language

- Declarative format
- Ruby Domain Specific Language (DSL)
- Resources: file, package, service, class ...
- Links: before, require; notify, subscribe

```
# install docker
class { 'docker':
  selinux_enabled => 'true',
  storage_driver  => 'devicemapper',
  service_state   => 'running',
  service_enable  => true,
  subscribe       => File['/etc/systemd/system/docker.service.d/startup.conf'],
}
```


Puppet is declarative: you define the end state of resources and their dependencies, not steps.

https://docs.puppet.com/puppet/latest/reference/lang_summary.html

In the example shown, nothing indicates *how* Docker will be installed, just that it should be installed. The implementation of the Docker module (class) is responsible for handling the installation based on the platform where the code is being run.

Puppet is written in a Ruby Domain Specific Language (DSL). Code is saved in files called Manifests, organized into Modules

Puppet has a resource abstraction layer which hides platform-specific details (most of them) so that you don't (usually) care whether you are working with this or that 'nix, Mac OS X or Windows

You define different Resources in Puppet:

- Files
- Packages
- Users
- Groups
- Services
- Cron jobs
- SSH keys

For example, the commands

```
puppet resource file /etc/passwd
puppet resource service ssh
```

will return the "Puppet picture" of these resources, which can be used to check the definition of the resource on a system, and ensure that it matches what is expected. For example, you can assert that a file must – or must not – exist, that it must contain certain information, and what the ownership and permissions must be. If the actual definition does not match the expected definition, the Puppet Agent will apply changes, creating (or deleting) a file with the expected contents and other attributes.

```
file {_____:
 ensure => directory,
 owner => _____,
 group => _____,
}
```

Also "Stages" can be used to enforce ordering/sequencing, although there are limitations to doing so.
https://docs.puppet.com/puppet/latest/reference/lang_run_stages.html

Puppet is considered "mostly" declarative since one could use "exec" resources to string together a set of commands to configure resources, bypassing the declarative model.

Package File Service Pattern (1)

- Most basic Puppet pattern:
  Package -> File -> Service
- Ensure a package is installed
- Deploy its configuration file(s)
- Ensure the service is running
- Set explicit dependencies

- Commonly used inside a module

The most basic pattern in Puppet is: "Package -> File -> Service"

- Package: ensures that a package is installed (installs it if it is not present)
- File: deploys configuration file(s)
- Service: ensures that the service is running properly

Links are established between the items so that a change to the configuration file will cause the service to restart.

Package File Service Pattern (2)

```
package { 'openssh-server':
  ensure => present,
}
->
file { '/etc/ssh/ssh_config':
  ensure => present,
  source => 'puppet:///modules/sansappsec/etc/ssh/ssh_config',
}
~>
service { 'sshd':
  ensure => running,
}
```

-> operator "ordering arrow": creates resource on left, then resource on right.

~> operator "notifying arrow": when resource on the left changes, notifies resource on right.

Note:

- The "->" operator is an "ordering arrow" – it will create the resource on the left, then the resource on the right
- The "~>" operator is the "notifying arrow" - when the resource on the left changes, it will notify the resource on the right.

With this simple set of rules, anytime /etc/ssh/ssh_config changes, Puppet will restart sshd.

Many modules build upon this basic pattern to provide a simplified interface to handle installing & configuring an application.

Writing Clean Puppet Code – Style Guides

Puppet publishes an official language style guide

- Readability, Simplicity, and Maintainability
  https://docs.puppet.com/guides/style_guide.html

Books, posts, and tools on best practices

- Puppet Best Practices book
- Puppet Cookbook project – example code and templates

Writing clean Puppet code helps ensure that your infrastructure can be maintained over the long run. Puppet has published a "Puppet Language Style Guide" (https://docs.puppet.com/guides/style_guide.html)in order to set the standard for how Puppet code should be written. By following the published standard, authors of Puppet modules make it easier for their code to be maintained, supported and understood.

Other resources on Puppet coding best practices:

https://puppet.com/blog/writing-great-modules-an-introduction

"Puppet Best Practices: Design patterns for maintainable code" by Chris Barbour
http://shop.oreilly.com/product/0636920038528.do

http://techblog.constantcontact.com/devops/treat-puppet-like-code/

https://davidmarquis.wordpress.com/2013/09/17/a-few-puppet-best-practices/

Puppet Cookbook project (http://www.puppetcookbook.com/) contains example code and templates.

Writing Clean Puppet Code – Tools

Many editors/IDEs have tools to assist with writing good Puppet code

- Atom, IDEA, Sublime, VIM, Visual Studio
- Geppetto – IDE designed for Puppet work

Lint checking

- puppet-lint for syntax/format
- puppet-lint-security - add-ons for security review

Eclipse IDE for Puppet development:

https://github.com/puppetlabs/geppetto/tree/dev/docs

https://github.com/puppetlabs/geppetto

http://puppetlabs.github.io/geppetto/

Puppet plug-in for IntelliJ IDEA, WebStorm:

https://plugins.jetbrains.com/plugin/7180?pr=

Puppet plug-in for NetBeans

http://plugins.netbeans.org/plugin/47067/puppet-configuration-editor

Puppet Lint for syntax checking – can be run in Continuous Integration. Includes an auto-fix capability for simple problems (using –fix argument). Puppet-Lint implements the official Puppet Style Guide. Plugins can be written to implement custom checks

http://puppet-lint.com/

http://bombasticmonkey.com/2014/08/19/puppet-lint-1.0.0/

Puppet Lint Security (security-specific checks that can be plugged in to Puppet Lint)

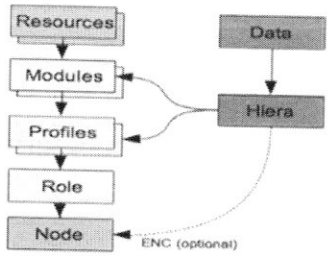
https://github.com/floek/puppet-lint-security-plugins

Writing Clean Puppet Code – Modules and Patterns (1)

Modules are used to organize code
Puppet Forge offers a wide variety of modules; review before use

Roles & Profiles pattern

- A node has exactly 1 role (business function)
- A role includes profiles to implement the function
- Profiles use modules to configure resources



Separate **Data** from **Code**... and encrypt sensitive data

Modules are bits of Puppet code, files, and other information bundled together to stand on their own. Since modules can include other modules, this provides a mechanism to build abstractions and relationships.

The "Roles and Profiles" pattern is a commonly used approach to configuring a node. Consider a basic web application server: some of the configuration may be different between dev, QA, and production, but the overall structure is the same. One could write a large web application server manifest, and look up configuration data with Hiera (a configuration data management tool included with Puppet). However, when the time comes to create a second, slightly different web application server, there will likely be large amounts of cut-and-paste, and drift, to build the second. Instead, with the roles and profile pattern, a new role is defined which re-uses many of the profiles and adds just the new application's needs.

The module hierarchy might look something like this:

```
role::application1 {
  include profile::base
  include profile::webserver
  include profile::appliction_one
}

role application2 {
 include profile::base
 include profile::webserver
 include profile::memcache
 include profile::application_two
}
```

Here you can see that the common base things for all servers are in one profile; common web server things in another, and a re-usable memcache profile is included for the second application's role. Now, when updates need to be made across all nodes, the base profile can be updated. From this point, all that's needed is to assign one role to a node and it will get configured as desired.

https://puppet.com/presentations/designing-puppet-rolesprofiles-pattern

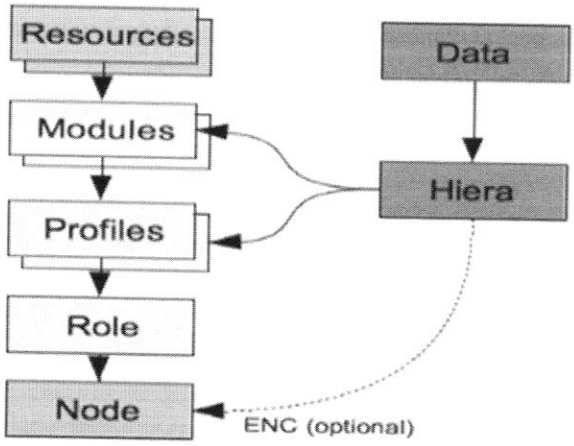
http://www.craigdunn.org/2012/05/239/

Writing Clean Puppet Code – Modules and Patterns (2)

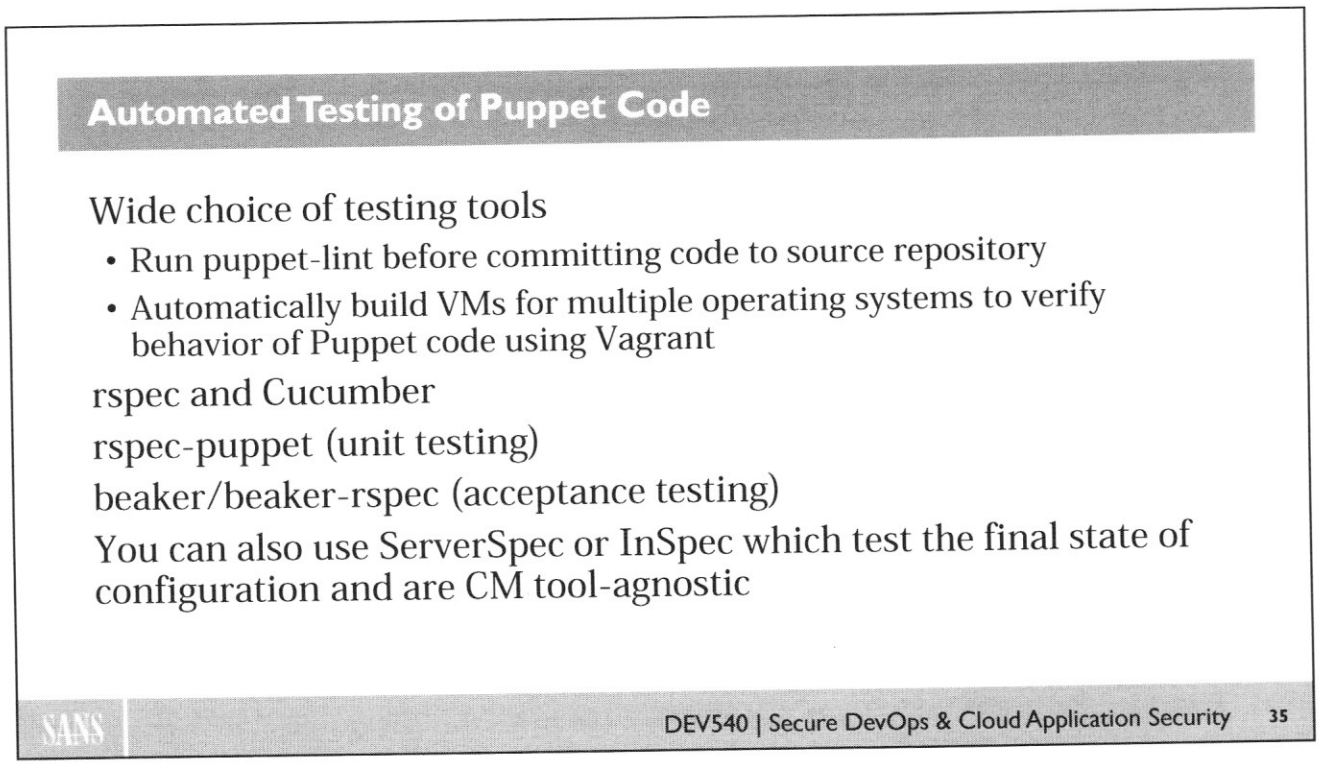
Modules configure a collection of resources (postfix server – package, files, service config)

Profiles use a set of modules to define a desired behavior (SMTP relay)

Roles specify which behaviors a node should have



This page intentionally left blank.



Testing frameworks can be used to write automated tests and checks for Puppet code, which can be run in Continuous Integration.

Tim Sharpe, author of Puppet-Lint and rspec-puppet, explains how to test Puppet modules:
http://bombasticmonkey.com/2011/11/04/test-your-puppet-modules-functions/

Puppet code is written in Ruby. You can use Ruby test frameworks (rspec and Cucumber) to write automated tests for Puppet manifests. However, newer testing frameworks make it easier to write tests for Puppet code.

rspec-puppet
An extension of the open source Ruby rspec BDD testing framework (http://rspec.info/)
http://rspec-puppet.com/tutorial/
https://github.com/rodjek/rspec-puppet
http://www.jedi.be/blog/2011/12/05/puppet-unit-testing-like-a-pro/

beaker and beaker-rspec
Puppet acceptance test framework
https://puppet.com/presentations/workshop-know-you-push-go-using-beaker-acceptance-test-framework-test-your-puppet
https://github.com/puppetlabs/beaker-rspec
https://github.com/puppetlabs/beaker

Security with Puppet - Basics

- Puppet-lint-security-plugins
- Use Hiera to separate configuration data, especially environment-specific data (including passwords, certificates, paths, etc.) from Puppet code that works on data
  - Be careful storing Hiera data files in source repo unless they are encrypted
  - Use hiera-eyaml or a secrets manager for sensitive data
- Don't use autosign for clients
  - This allows one node to impersonate another

Puppet-lint-security-plugins. A set of puppet lint checks for security good practices:
https://github.com/floek/puppet-lint-security-plugins

Separate data in Puppet from code that works on data, using Hiera: a key/value configuration data management tool (https://docs.puppet.com/hiera/)

Store passwords and other credentials in Puppet Hiera using hiera-eyaml (https://github.com/TomPoulton/hiera-eyaml) or use a secrets manager

https://ask.puppet.com/question/476/securely-storing-passwords-and-keys/

http://serverfault.com/questions/376613/making-puppet-manifests-modules-available-to-a-wide-audience

https://puppet.com/blog/encrypt-your-data-using-hiera-eyaml

Don't enable autosign for Puppet nodes, especially in production

https://docs.puppet.com/puppet/latest/reference/ssl_autosign.html

Security with Puppet – Auditing and Reporting

Audit mode for detective change control

- Puppet noop
- Does not change configuration
- Reports variances between environment and manifests

Reporting

- Create custom reports from PuppetDB
- E.g. All systems with a vulnerable version of a package (e.g. OpenSSL 1.0.2e)
- Corrective vs. Intentional changes

Puppet provides a NOOP Option to run in audit mode: check and report on variances between system config and manifests. This provides detective change control and can be run as a pre-check before applying changes (to verify what systems will be impacted).

All changes made by Puppet are audited in the Puppet logs.

You can use the centrally-managed information in the Puppet Master and PuppetDB to create reports on configuration state

Puppet versions greater than 4.6.0 include the ability to detect corrective changes – for example, when puppet reverts a change made directly on the system.

Puppet Forge modules tagged compliance: https://forge.puppet.com/tags/compliance

Security (and Compliance) with Puppet – SIMP

Systems Integrity Management Platform (https://simp-project.com/)

- Comprehensive set of Puppet modules to build/harden RHEL and CentOS systems to compliance standards (Windows is under development)
- Extensive automated test suite to make policies safe to apply
- Open source collaboration with NSA
- Focused on meeting NIST 800-53, FIPS 140-2, DISA STIG, SCAP Security Guide
- Includes compliance mapper to create reports
- Enterprise assistance and support available from OnyxPoint

SIMP – Systems Integrity Management Platform ((https://simp-project.com/)) is an extensive set of open source Puppet modules (with automated tests) to enforce security compliance with Security Content Automation Program (SCAP) profiles for RHEL and Centos 6 and 7 platforms. Extensively tested to increase confidence that hardening policies can be applied without breaking system behavior.

OnyxPoint (https://www.onyxpoint.com/), the maintainer of SIMP, provides commercial support and assistance

Announcement: https://puppet.com/blog/nsa-release-of-system-integrity-management-platform-simp

Documentation: http://simp.readthedocs.org/en/latest/index.html

Puppet Modules:

https://github.com/NationalSecurityAgency/SIMP

https://github.com/simp

https://forge.puppet.com/simp

https://software.forge.mil/sf/projects/simp

Watch: "A Year in Open Source Automated Compliance with Puppet – Trevor Vaughan", PuppetConf 2016
https://www.youtube.com/watch?v=a270uDh8muE

Lab #2.1 – Manage Configuration with Puppet

Duration Time: 30 Minutes

Use Puppet to install AIDE on all of the "workstations" in our environment.

OBJECTIVES

See puppet role / profile / module structure in action
Install & configure an application on a group of machines

PREPARATION

Check out puppet configuration information from GitLab.

This page intentionally left blank.

Lab #2.1 Summary – Manage Configuration with Puppet

In this lab, you...

- Reviewed the basics of how the class VM is built
- Used Puppet to deploy software to all nodes with a specific profile
- Used our CI/CD pipeline to validate Infrastructure Code
- Used Git hooks to automate simple tasks

In this lab, we've looked into the basics of how the class VM is built, and how Puppet can be used to deploy software to a group of nodes. While this was a highly simplified example, it is still clear that centralized configuration management provides immense power.

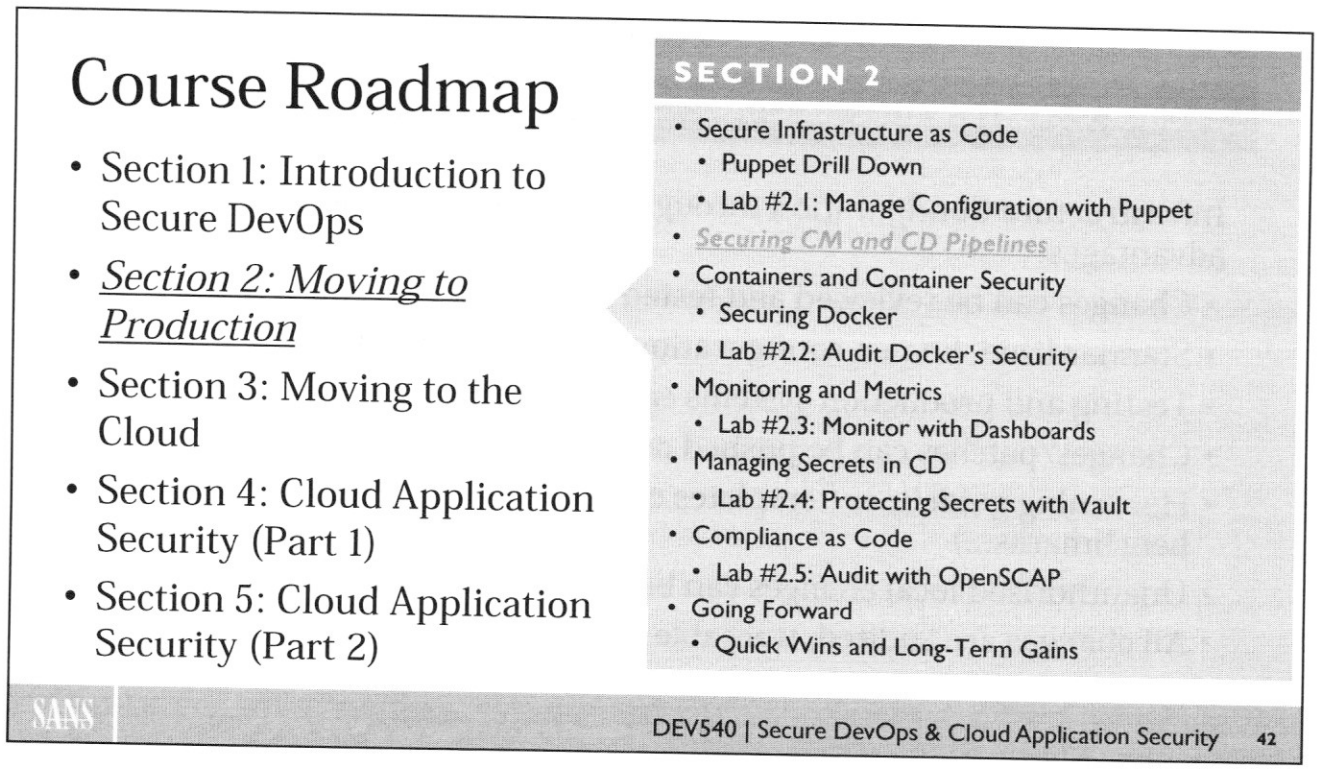
Along the way, we also reviewed ways we can take advantage of our CI/CD pipeline to validate our Infrastructure Code and ways we can use Git hooks to automate simple tasks.

Secure Infrastructure as Code - Summary

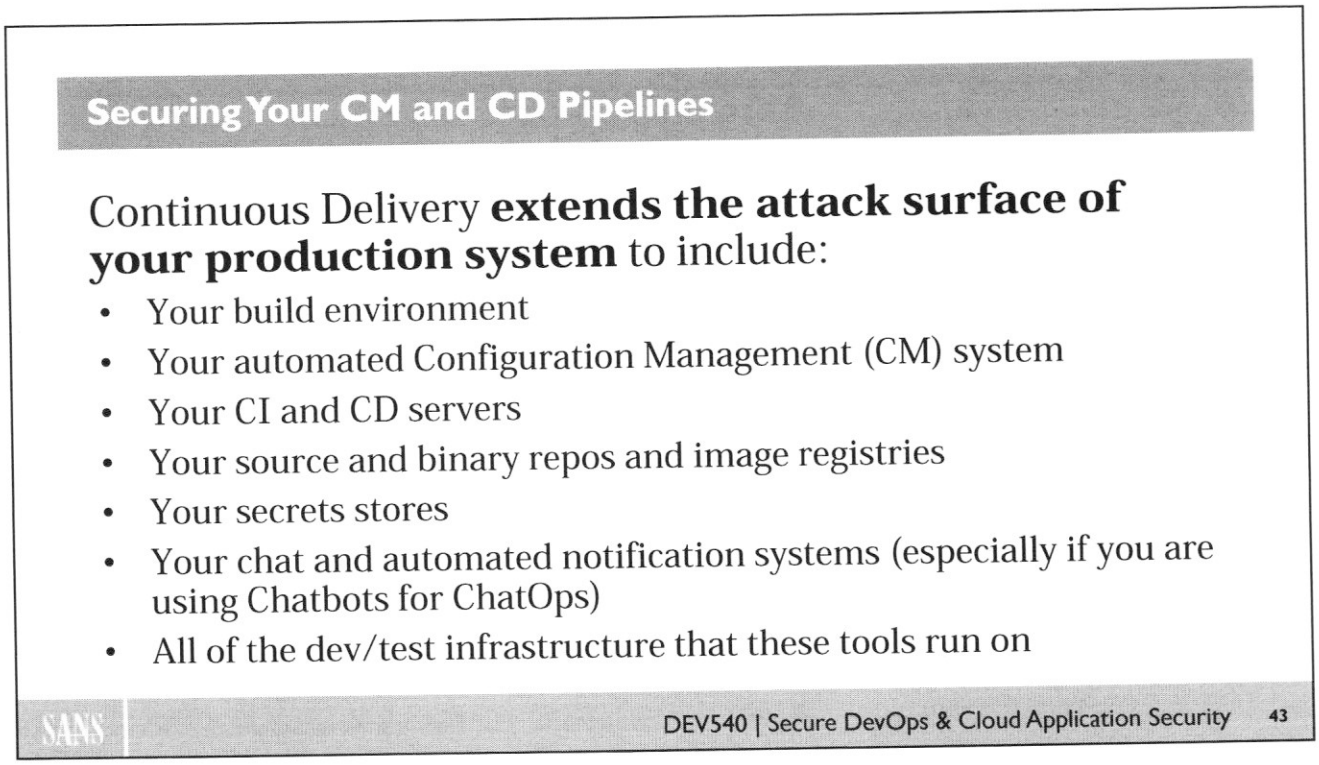
Infrastructure Configuration through Code provides several major advantages:

- Changes can be reviewed and tested in advance, to reduce risks
- Standardized, known configurations (no snowflakes)
- Testing and production systems can be aligned
- Changes/patches can be pushed out quickly and safely
- Hardening guidelines/templates can be applied upfront (STIGs, CIS benchmarks...)
- Unauthorized local changes can be caught and easily corrected
- All changes are audited, traceable and testable

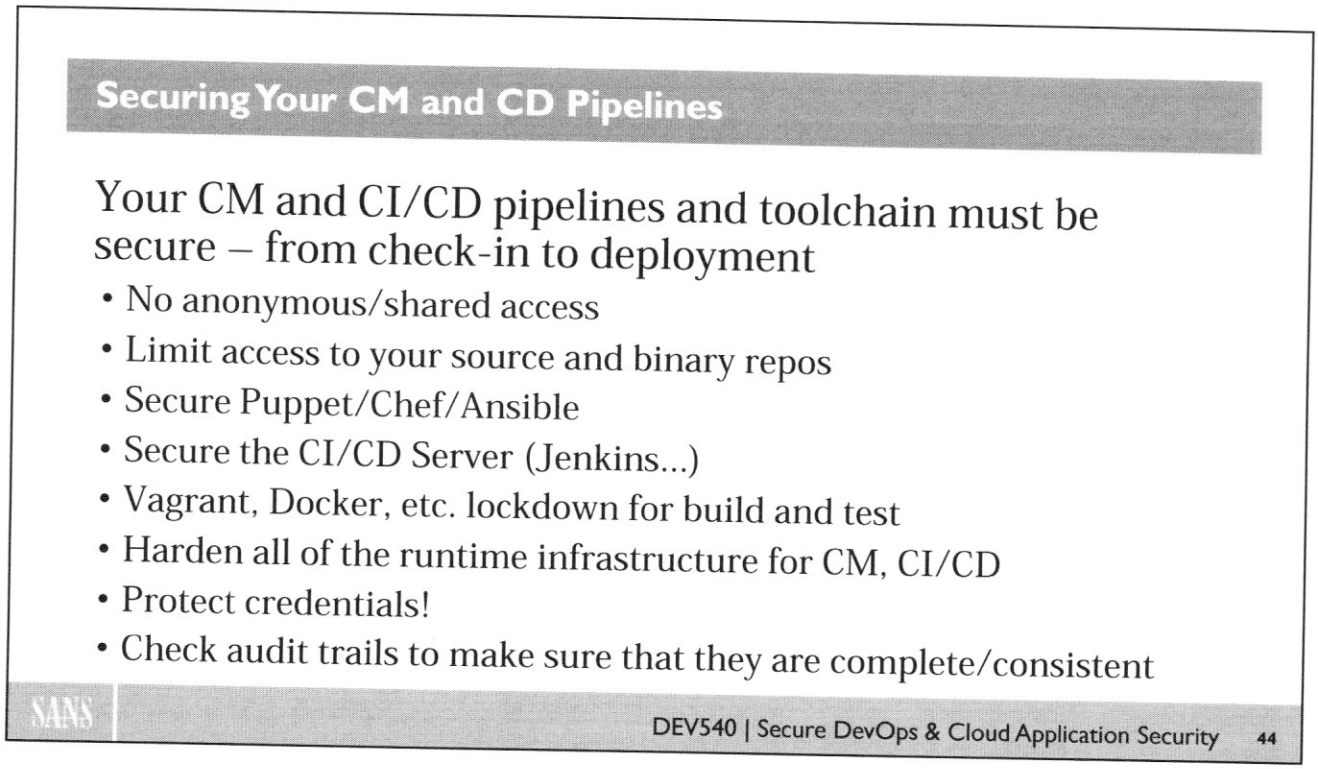
This page intentionally left blank.



This page intentionally left blank.



Continuous Delivery and Continuous Deployment extend the attack surface of your production systems to include all of the tooling and source and artifact repositories that you use to build and deploy systems. These systems (and the infrastructure that they run on) should be hardened, managed and monitored in the same way that you harden, manage and monitor your production systems.



CI/CD environments are generally set up for engineering ease of use, efficiency, speed, and convenience. This means that they may not be (read: probably aren't) secure.

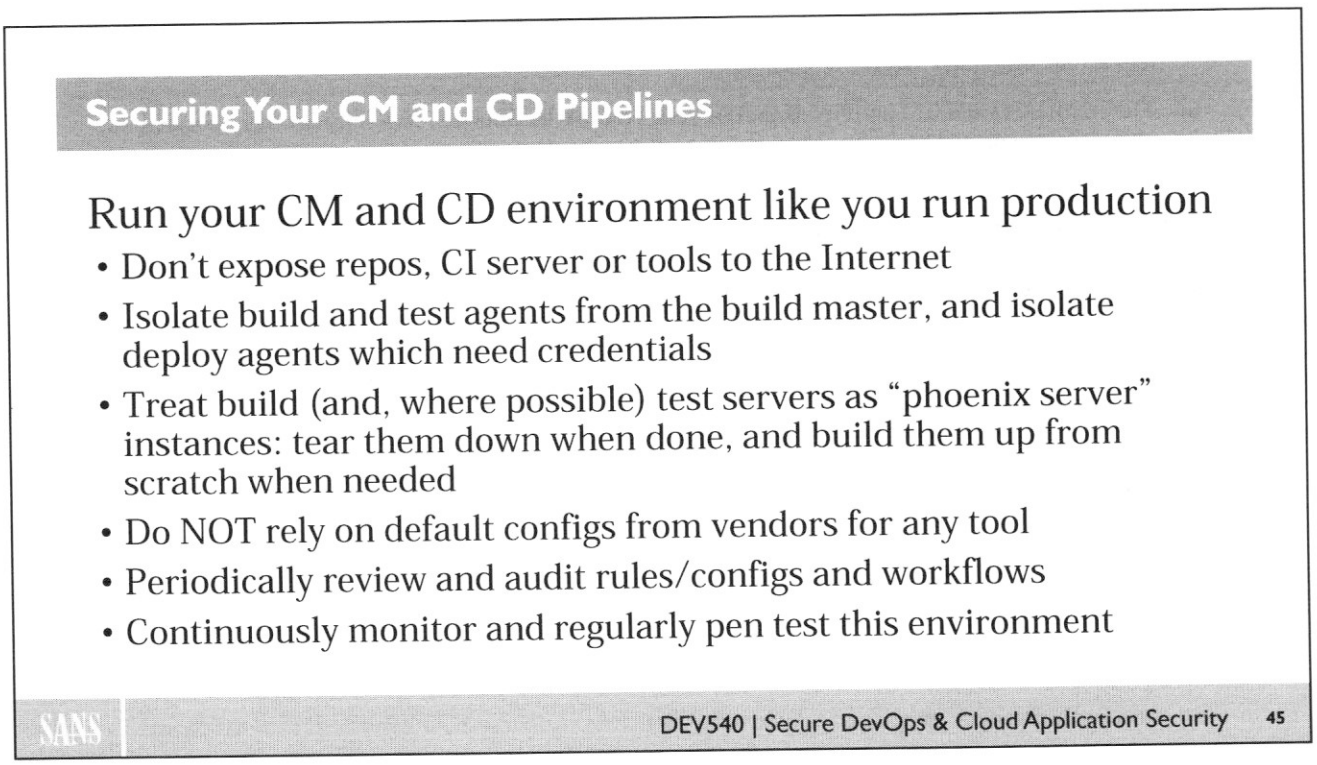
At the Black Hat conference in 2015, security researcher Nikhil Mittal presented research on why "Continuous Integration tools are a hacker's best friend": https://www.blackhat.com/docs/eu-15/materials/eu-15-Mittal-Continuous-Intrusion-Why-CI-Tools-Are-An-Attackers-Best-Friend.pdf. Mittal found serious, exploitable vulnerabilities in 5 different CI/CD servers: Jenkins, CruiseControl, Go, TeamCity, and Hudson.

> "Running a poorly configured CI tool is like providing a ready-to-use botnet for anyone to exploit."

The CI/CD pipeline provides a map and a path to production. If compromised, it can provide an attacker with access to the build environment and source code, and high-privilege access to systems across development, test, and potentially production.

You need to minimize the chance of compromise and to minimize the risk if any of the components are compromised. The integrity of the CD pipeline, its runtime environment, and the artifacts created need to be ensured. It is also important to detect and prevent insider attacks against the CD environment.

This is especially important if you are using a cloud-based code repository (e.g. GitHub) and/or cloud-based build environment.



Some basic steps to take:

1. Make sure that you do not expose your repos, CI server or other tools to the Internet as publicly available – see "Why compromised Jenkins can lead to a disaster" http://www.tothenew.com/blog/why-compromised-jenkins-can-lead-to-a-disaster/
2. Harden the CI/CD environment and pen test it
3. Keep patches up to date (for the OS and toolchain). Carefully vet and test all plugins and upgrades to your tools.
4. Isolate build and test agents from the build master, and from deploy agents which need credentials to systems. Use Docker or other containers or VMs to isolate test agents, especially if you are sharing test infrastructure with other systems (private or public cloud). Remember that tests are code, and any code could be malware: for example, a malicious insider could write tests to exfiltrate credentials.
5. Treat build and test servers as Phoenix Server instances: build them when you need them, tear them down when you are done. This reduces your attack surface and it ensures that your environment is always in a known, reproducible state.
6. Isolate development and test environments from production – make sure that it is not possible to accidentally reference a production system from dev/test
7. Enforce authentication and access control rules for CI and for deployment. Audit trails are not reliable if you can't ensure the identity of the people who performed actions. Prevent anonymous access to the repositories or the CI server, and do not allow people to share user accounts.
8. If you grant sudo or ssh access to users for production access, make sure to limit the commands that they can perform.
9. Review changes to all CD pipeline configuration/rules and periodically trace through the full cycle and audit trails
10. Checksum all artifacts and scan artifacts for malware as part of the CD pipeline

11. Protect all credentials used to access development, test, and production systems and resources. See the section on Managing Secrets in this course
12. Monitor CI/CD and CM activity as part of your security monitoring, watching for unauthorized changes, unexpected access to repos and to secrets

DO NOT use a Continuous Integration server configuration out of the box, especially if you want to implement Continuous Delivery or Continuous Deployment. By default, Jenkins, for example, does not require authentication, runs with high privileges and is easily hacked.

Follow these steps for securing/hardening the Jenkins CI server: https://wiki.jenkins-ci.org/display/JENKINS/Securing+Jenkins

Securing your Puppet Configuration Management Toolchain

Puppet (or Chef or Ansible...) becomes a prime target of attack

- Master/repo holds configuration information for *everything*
- CM toolchain is a critical point of failure for operations management
- Adversaries can use this to compromise systems, steal information or credentials, or to take control over your production operations
- Also, need to watch out for malicious insiders

Harden infrastructure and carefully restrict access to Master

Ensure that Puppet runtime and configs are safely backed up

Review and test updates and community recipes carefully

Track and patch Puppet vulnerabilities

Your central configuration management tool and configuration database is a natural target of attack/compromise for bad guys and insiders. Puppet is based on a strong master/agent architecture. The agent provides facts to the master and waits for instructions. If the Puppet Master or Server is compromised, it can be used to make any changes to any systems that are under management.

According to Puppet, the default configuration for Puppet is secure. However, there are some basic hardening steps that you should take:

- Harden all of the infrastructure
- Scan/pen test to ensure that the control environment is secure
- Restrict access to the Puppet Master/Server (host firewall....)
- Ensure that Puppet code is up to date
- Review and test all community manifests/recipes – don't just accept them and run them
- Continuously review your Puppet logs

Some additional steps that you may want to take to harden the configuration - see chapter 5 of "Learning Puppet Security" book by Jason Slagle http://www.amazon.com/Learning-Puppet-Security-Jason-Slagle/dp/178439775X

Puppet uses HTTPS between agents and the Server/Master. It has a built-in CA. Puppet can be configured to use an external CA

https://docs.puppet.com/puppet/latest/reference/subsystem_agent_master_comm.html

https://docs.puppet.com/puppet/latest/reference/config_ssl_external_ca.html

Track Puppet security announcements (vulnerabilities and upgrades/patches): https://groups.google.com/forum/#!forum/puppet-security-announce

For more on securing your Configuration Management chain, read "Who Watches the Watchmen: Securing Configuration Management Systems" http://blog.threatstack.com/who-watches-the-watchmen-securing-configuration-management-systems

Securing your Chef environment

- Secure the Chef Server and Workstation(s)
- Carefully manage credentials and other secrets—more on this later
- Keep up with latest Chef and OS patches
- Define a secure workflow for promoting cookbooks to test and to production

"How to be a Secure Chef" (https://learn.chef.io/skills/be-a-secure-chef/) outlines the basic steps for securing Chef configurations

Secure the Chef Server

- Use a strong password for the admin account
- Log SSH connections
- Use PKI
- Use firewalls
- Protect port 443
- Secure the management console
- Disable root login
- Install latest security patches
- Use Chef Analytics to monitor security-related objects (new organizations, new nodes, uploading/removing cookbooks)
- Use RBAC
- Follow the principle of least privilege
- Use Chef organizations (top-level structure for RBAC)

Manage Chef Secrets

- Encrypt sensitive data – Search recipes, templates and data bags for plaintext secrets. Use encrypted data bags and Chef Vault.

Secure the Chef client (https://docs.chef.io/chef_client_security.html)

- Install latest security patches
- Use validatorless bootstrapping

Securing the Chef workstation

- Keep up with latest OS patches
- Use strong passwords
- Log off if unattended
- Place user keys in a secure location

Securing the CD Pipeline - Example

Rotten Apple project

A Ruby example that shows how to audit/attack a CI/CD environment – although it is no longer active, you can use it as a starting point to build your own audit/checker

Sample attacks:

- Steal API key or SSH private keys
- Flush IP tables (drop firewall rules)
- Install attack tools
- Make an unauthorized commit to master
- Execute an Nmap scan
- Break out to a command shell

Rotten Apple is an example of a project for checking/testing Continuous Integration (CI) or Continuous Delivery (CD) system security. https://github.com/claudijd/rotten_apple

https://www.trustwave.com/Resources/SpiderLabs-Blog/Securing-Continuous-Integration-Services/

RottenApple::Audit

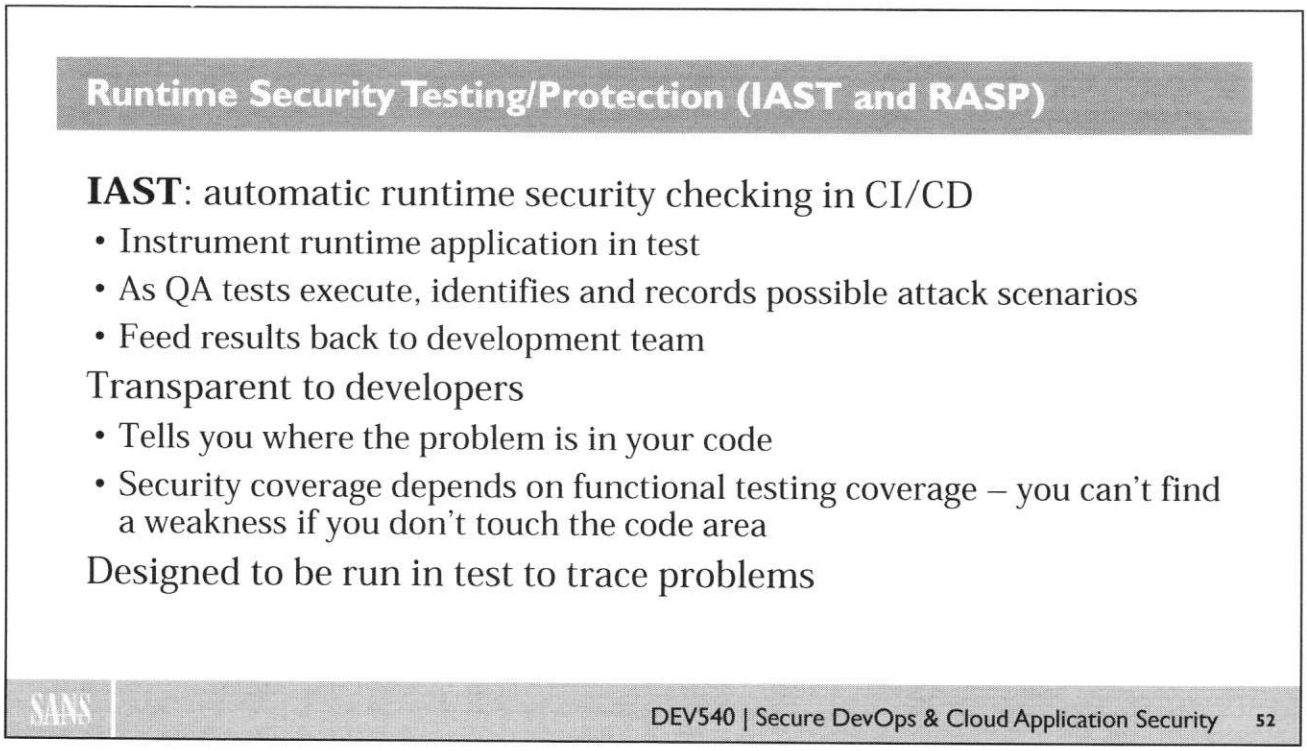
This portion of the project is focused on auditing a CI/CD system and is the default name space when the "rake" command is invoked in this project. It performs the following audit checks:

- Is the root user is being to build projects?
- Can malicious code steal your RubyGems API key?
- Could malicious code pivot to private networks?
- Can malicious code authenticate using your GitHub creds?
- Could malicious code receive instructions from a remote party or exfiltrate data from your CI?
- Can malicious code access other projects being built on the same server?
- Can malicious code steal SSH private keys?

RottenApple::Attack

Conversely, this is the portion of the project that enables you to actively attack a CI/CD. To change to "attack mode", open the Rakefile and change the default to "attack". List of attacks:

- Steal the RubyGems API key
- Flush IP Tables (aka drop firewall rules)
- Install Software to aid in the attack process
- Make an unauthorized commit to master
- Perform an Nmap scan of a desired set to targets
- Throw/Shovel a reverse shell to get command-line access to the CI/CD
- Steal SSH private keys



RASP (Runtime Application Security/Self-Protection) and IAST (Interactive Application Security Testing) solutions instrument the runtime environment such as the Java JVM or the .NET CLR and passively track (IAST) – and potentially block (RASP) – security vulnerabilities that could be exploited at runtime. These solutions can identify and report vulnerabilities down to the specific line of code, with full call trace/data flow analysis.

IAST (Interactive Application Security Testing)

IAST tools can run in CI/CD, generally as part of automated functional acceptance/integration testing. For example, as a Selenium acceptance test suite executes against the application, the IAST runtime will monitor the application and catch problems using taint analysis and signature checking – this is similar to running the tests through a passive attack proxy like ZAP. Exceptions are tracked, and can be reported back to the CI/CD server dashboard, bug tracking systems etc. (through API integration).

IAST tools don't fuzz files, fields or APIs, or attempt other kinds of security attacks – they passively watch code as it is executed and look for security vulnerabilities. This means that the quality of security test coverage will depend on the quality of your automated functional test coverage. If your tests don't hit certain paths in the application, then the IAST runtime monitor won't be able to see problems in those parts of the code (unlike a SAST tool).

Runtime Security Testing/Protection (IAST and RASP)

RASP: automatic runtime security protection in production

- Instrument runtime application in production
- Automatically catches and blocks attacks as they occur at runtime
- Adds runtime overhead

Compensating control, similar to Web Application Firewalls (WAFs)

- Can be used to help secure legacy apps and third-party apps
- Can run in blocking mode or monitoring mode
- Provides transparency into attack activity
- Like WAFs, can apply specific signature-based vulnerability patches

RASP (Runtime Application Security/Self Protection)

RASP is implemented in production, to block known attacks, patch over known public vulnerabilities, and provide visibility into application-level attacks as they occur.

Like Web Application Firewalls (WAFs), these tools provide an operational solution to security weaknesses in an application, either as a compensating control (for vulnerable legacy applications) or as part of a defense-in-depth approach.

One of the arguments for RASP/runtime security testing tools is that DevOps teams following Continuous Delivery/Deployment move too fast for InfoSec to keep up with testing and reviews. It may make more sense to build more security protection into the runtime environment instead of trying to build security into the SDLC and deployment processes.

RASP in production adds runtime overheads, and, like WAFs, there is the risk of false positives, where valid actions may be blocked. Most implementations of RASP (like WAFs) start with monitoring attacks, not blocking them, until operations gain confidence.

In monitoring mode, RASP solutions can provide insight and transparency into runtime attacks. This information can be fed into log management systems like Splunk, or into SIEM systems.

For more on the benefits of and future potential for RASP, see Gartner's "Maverick Research" report: "Stop Protecting your Apps; it's time for Apps to Protect Themselves", by Joseph Felman, September 2014

https://www.gartner.com/doc/2856020/maverick-research-stop-protecting-apps

http://www.computerweekly.com/opinion/Security-Think-Tank-RASP-a-must-have-security-technology

RASP and IAST Solutions

Commercial Solution	Supported Environments
Contrast Security	Java, Node.JS, .NET
HP Application Defender	Java, .NET
Immunio	Ruby on Rails, Python, Java, Node.JS
Prevoty	Java, .NET, Ruby, PHP, Python, Node.JS, Go
Seeker	Java (+Scala, Groovy, Clojure), .NET
Waratek	Java

There are only a small number of commercial IAST/RASP solution providers today, each working in different ways:

Contrast (http://www.contrastsecurity.com/features)

Provides runtime RASP (blocking) and IAST (runtime vulnerability tracing) solutions for Java (using the Java Instrumentation API), Node.JS, and .NET (Visual Basic, C#) applications. A Contrast agent must be run on every system that is under its protection. These agents receive rules and report vulnerabilities and other information back to a central server (on premise or SaaS). Contrast identifies vulnerabilities in running code, including third party/open source libraries, and will automatically inventory applications and libraries and identify libraries that are out of date or that contain known vulnerabilities (CVEs). You can query the inventory to find all applications that use a specific library, and apply CVE shields to report on exploits of specific vulnerabilities or to block them.

In RASP mode, Contrast provides visibility into attacks as they occur, including which applications are being attacked, where they are being attacked from, and whether the attacks are successful. You can blacklist IPs at the application level, or create virtual patches to filter out attack signatures on the fly. Contrast can be implemented into CI/CD using a REST API. There is also a Contrast plug-in for Jenkins: https://github.com/Contrast-Security-OSS/contrast-jenkins-plugin

HP Application Defender (https://saas.hpe.com/en-us/software/application-defender)

HP Application Defender provides runtime security threat detection and protection for Java and .NET applications. Agents running on each system report back to a central cloud-based SaaS dashboard. Results can also be exported into XML, JSON or plain-text format to import into bug tracking systems. Application Defender is based on runtime security technology first developed at HP back in 2008.

See 451 Research review of HP Application Defender, November 2014: https://451research.com/report-short?entityId=83291

Immunio (https://www.immun.io/)

Immunio is a startup that offers SaaS runtime protection for web applications developed in Java, Ruby on Rails and Python. According to the vendor, it offers protection against SQL injection, cross-site scripting, and remote command execution attacks, and it will blacklist known attackers.

Prevoty (https://prevoty.com/)

Prevoty provides runtime application security monitoring (ASM) and application security protection (ASP) solutions. Applications are instrumented using plug-ins for Java web (via servlet filter) or .NET (through an HTTP module DLL for IIS) web apps (no application code changes required), or by making calls in the application to Prevoty's API to forward payloads to a Prevoty Security Engine for analysis. Malicious payloads are neutralized (on input or output) and sanitized content is returned to the application in milliseconds (latency dependent on whether the engine is on premise or in the cloud). Prevoty blocks XSS and SQL injection and command injection attacks, and session theft and CSRF and man-in-the-middle attacks by automatically issuing and managing session tokens.

According to Prevoty, they are the only RASP provider that implements LANGSEC – advanced language-specific security protection.

http://www.csoonline.com/article/3033917/application-development/is-language-theoretic-security-the-answer-to-internet-insecurity.html

A limited-edition of Prevoty ASM (monitoring) is available for free: Cloud-based monitoring of Java (only) applications for XSS (only) attacks

http://blog.prevoty.com/changing-the-application-security-game

http://info.prevoty.com/asm-basic

Seeker (http://www.coverity.com/products/seeker/)

IAST technology acquired in 2014 from Quotium, for Java, Scala, Groovy, Clojure, .NET (C#, VB.Net), PHP, PL-SQL, T-SQL

http://www.quotium.com/resources/automating-agile-code-security-testing-quotium-seeker-selenium-scripts/

Waratek (http://www.waratek.com/)

Run-time protection for Java applications, implemented inside a separate secure Oracle HotSpot JVM which wraps the application runtime (a "secure container"). It includes protection for common attacks like SQL injection (against popular SQL databases), known vulnerabilities in the JDK and common Java libraries, and checking for proper file and network access. Waratek AppSecurity can take the results of DAST/SAST scanners and create new rules for any vulnerabilities found in testing the application ("virtual patching"). Waratek advertises zero false positives and zero false negatives.

Deutsche Bank case study: http://www.waratek.com/news/how-a-major-bank-hacked-its-java-security/

Comparing RASP to WAFs

- RASP does not rely on perimeter security model—you secure each runtime container
- RASP is easier to implement
- RASP can catch problems that WAFs can't see
- RASP has fewer false positives than WAFs
- Both RASP and WAFs support virtual patching
- RASP is limited to specific runtime platforms
- RASP adds runtime overhead to the application

RASP technologies are often compared to Web Application Firewalls (WAFs). One fundamental difference between RASP and WAFs is that RASP does not rely on a perimeter-based security model, where all traffic is routed through a firewall – the basis on which WAFs operate.

WAFs filter inbound (and sometimes outbound) traffic, and use pattern-matching or signature-matching algorithms to detect possible attacks – and potentially block them – if the WAF is configured inline to traffic and in blocking mode. RASP solutions secure each application runtime, which makes them better fitted for cloud and microservices architectures.

Read "Protection from the Inside : Application Security Methodologies Compared" (an analysis of RASP and WAF technologies by SANS Analyst Jacob Williams, April 2015): https://www.sans.org/reading-room/whitepapers/application/protection-inside-application-security-methodologies-compared-35917 This report compared a leading RASP solution (HP Application Defender) with an unnamed commercial WAF solution. The analysis found:

- RASP is easier to implement and requires less ongoing review/tuning
- RASP solutions catch runtime errors and unhandled exceptions and data leaks (for example, logging of confidential/private data) and application configuration problems which are not accessible to WAFs
- RASP have fewer false positives and are not subject to the same kinds of evasion techniques that attackers use against WAFs
- WAFs are not limited to specific languages/runtimes (most RASP solutions work only with Java and/or .NET managed code)
- Both WAFs and RASP can support virtual patching—taking the results of a static or dynamic scan, and using this to build rules for RASP to block specific vulnerabilities, reducing the window of exposure

Also: "RASP vs WAF – 5 Reasons to pick RASP" by Sharon Solomon July 6, 2015: http://securitybloggersnetwork.com/2015/07/rasp-vs-waf-5-reasons-to-pick-rasp/

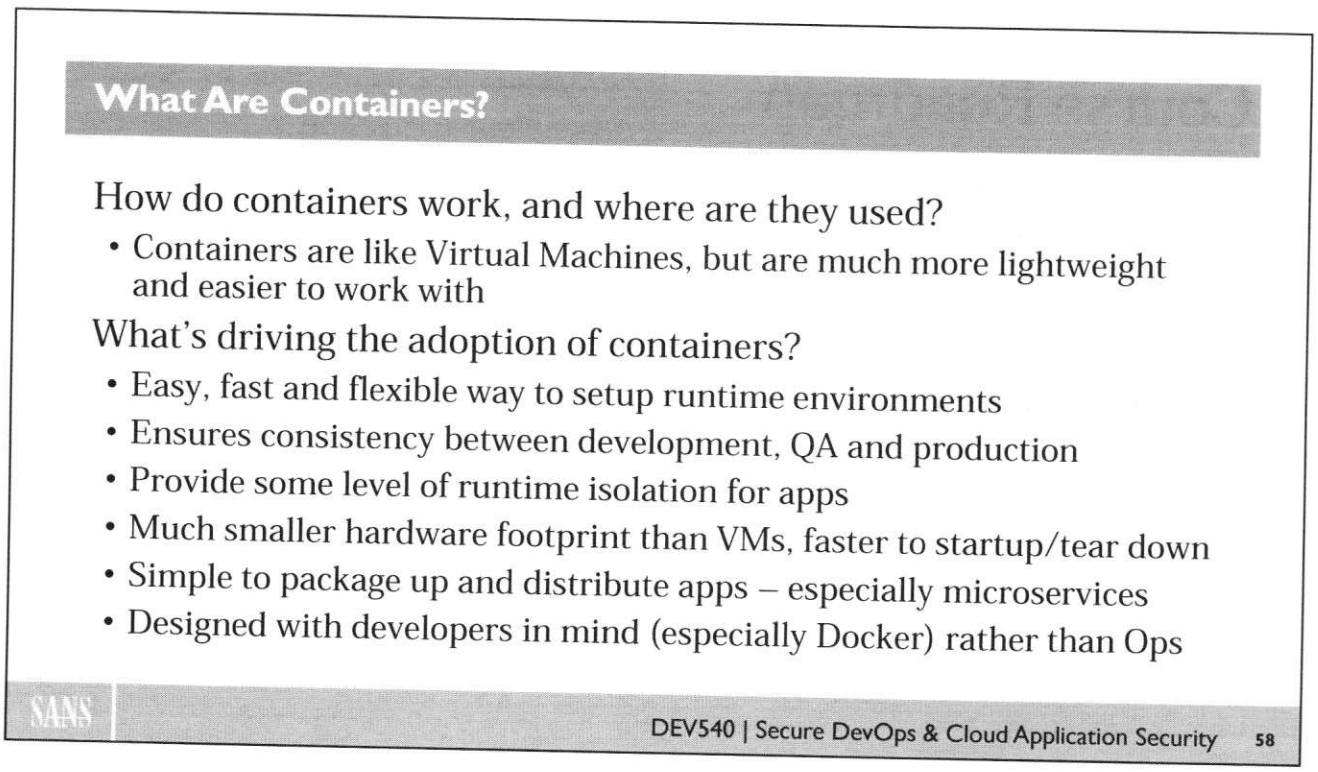
Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- *Containers and Container Security*
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.



Containers are isolated user spaces that share the same Linux OS kernel. Containers behave like virtual machines. Application dependencies are contained, making it possible to run applications with conflicting dependencies (e.g., different versions of the same library) on the same host.

However, containers are much more lightweight that VMs: there is no hypervisor, so they require much less CPU and RAM. Containers are cheaper to set up and run, they can take only a few seconds to provision and less than a second to startup or shutdown, and you can run many more of them on a physical server (in some cases, 100s of containers on one physical server). They offer more isolation than running multiple apps on bare metal physical servers, but with much less runtime footprint than VMs.

Containers provide a cheap and fast way to set up and tear down development and testing environments that are constantly changing, and an easy and repeatable way for developers to build up their own runtime environments without the help of a sys admin. With a few commands, developers can download pre-built/pre-defined images of common runtimes (a database like Postgres or a HTTP server like nginx, on top of a base OS image), and easily package up their application and everything that it needs to run (the app and all of the runtime dependencies including libraries and system tools) into a single image file. This image can be uploaded to a repository, making it easy to review, copy and share, and then push to test and to production.

Containers naturally fit into DevOps workflows, significantly reducing the cost, complexity and time needed to package and deploy applications. A developer can easily create an image of the application runtime environment that they need on their laptop or desktop and then push it to a Linux server, a VM or a cloud service like AWS or Azure. Using containers ensures consistency between development and production because the application runtime is set-up exactly the same in development, test and production.

Containers (especially Docker) are intended to run single applications – this makes them especially well-suited to a microservices-based architecture.

Containers - Caveats

Emerging technology, changing rapidly

- Lots of work being done by community and by vendors, including startups
- Development toolchains and operations ecosystem are still works in progress
- Security situation is continuously changing
- If your organization is using containers, you need to watch what's happening closely and keep up with the latest changes and guidance

The world of containers, especially Docker, is changing quickly. Major changes are being announced every few months, especially around packaging, operations management, and security capabilities and tooling. Make sure to keep up with the latest changes and guidance on container security and operations.

Container Options

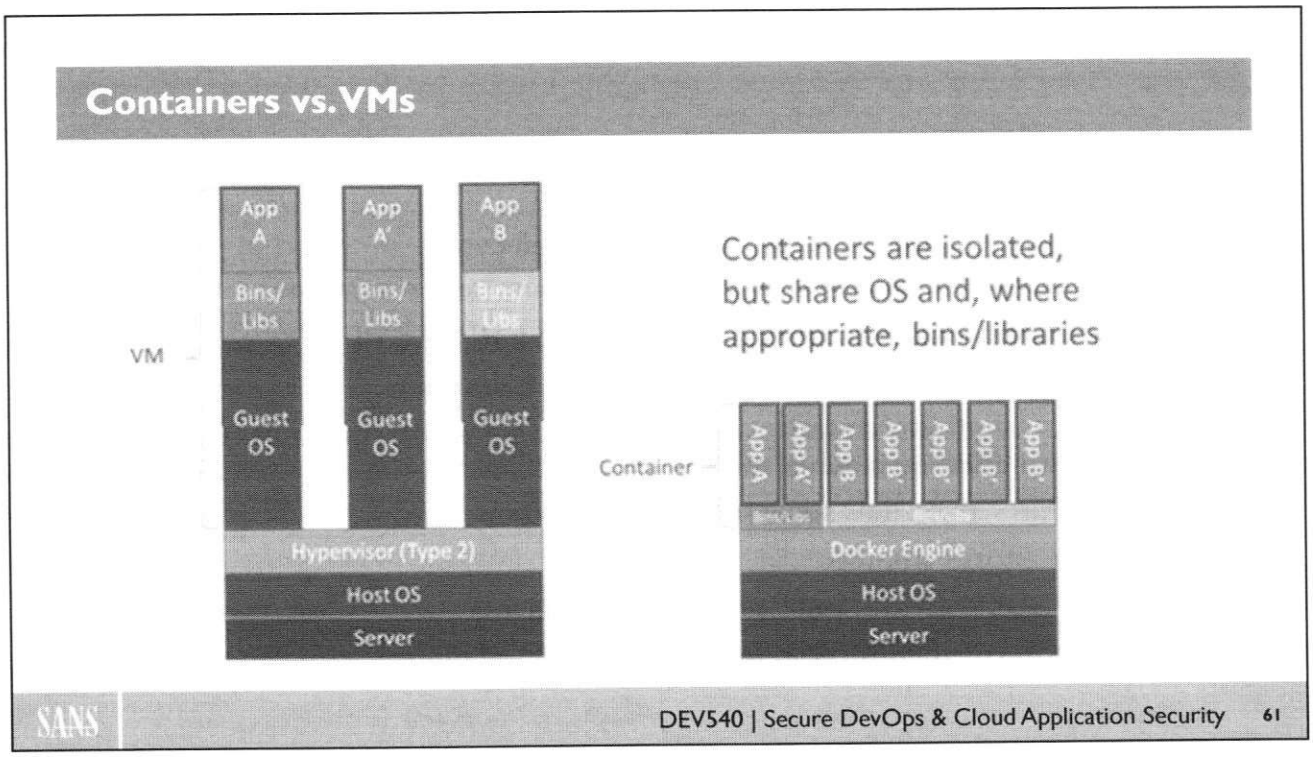
Container	Description
Linux LXC	Native capabilities in Linux, developed by IBM and Google Implement user spacing on top of cgroups and namespaces
LMCTFY	Open Source version of Google's internal container stack Discontinued – Google is working with Docker community and OCI on runC
rkt	CoreOS pluggable alternative to Docker Designed to be more extensible and more secure by default than Docker
LXD	Ubuntu extensions to LXC to create a pure-container hypervisor for Linux apps
runC	Lightweight runtime container based on OCI specification, created by Docker
Docker	Portable application container built on top of runC The 800-lb. gorilla of containers – massive industry adoption and momentum **This course will focus on Docker**

Leading container options (all of them Open Source) include:

- Linux **LXC** (https://linuxcontainers.org/) \Original implementation of user spacing in Linux on top of cgroups and namespace isolation (based on work done by IBM and Google)
- LMCTFY – Let Me Contain That For You (https://github.com/google/lmctfy) is the open source version of Google's internal container stack. It has been discontinued as Google collaborates with Docker on runc
- **Docker** (https://www.docker.com/what-docker) is by far the most popular container solution. There are over 100,000 "Dockerized" applications in the Docker Hub registry. Docker provides easy-to-use packaging and other tooling aimed at developers. It was originally implemented on top of LXC. Later (in Docker 0.9) Docker created its own runtime container originally libcontainer (https://github.com/docker/libcontainer) and later runc (https://github.com/opencontainers/runc)
- **runC** (https://runc.io/) is the underlying runtime for containers developed by and used by Docker and implementing the Open Container Initiative (OCI) specification. It is built on top of libcontainer
- CoreOS **rkt** (https://coreos.com/rkt/ – A pluggable alternative to Docker, designed to address what rkt authors see as some fundamental shortcomings in the Docker architecture. Designed to be more open and secure than earlier versions of Docker.
- Ubuntu **LXD** (https://www.ubuntu.com/cloud/lxd) A pure-container hypervisor to run Linux Oses and applications

The history of container development is nicely presented here: http://pivotal.io/platform/infographic/moments-in-container-history

Much arguing over the direction of containers and concern over Docker's dominance has led to standardization initiatives especially the **Open Container Initiative (OCI)** https://www.opencontainers.org/ (which Docker and CoreOS and Google, IBM and Red Hat and others have joined) to agree on interchangeable image formats and documented runtime specifications.



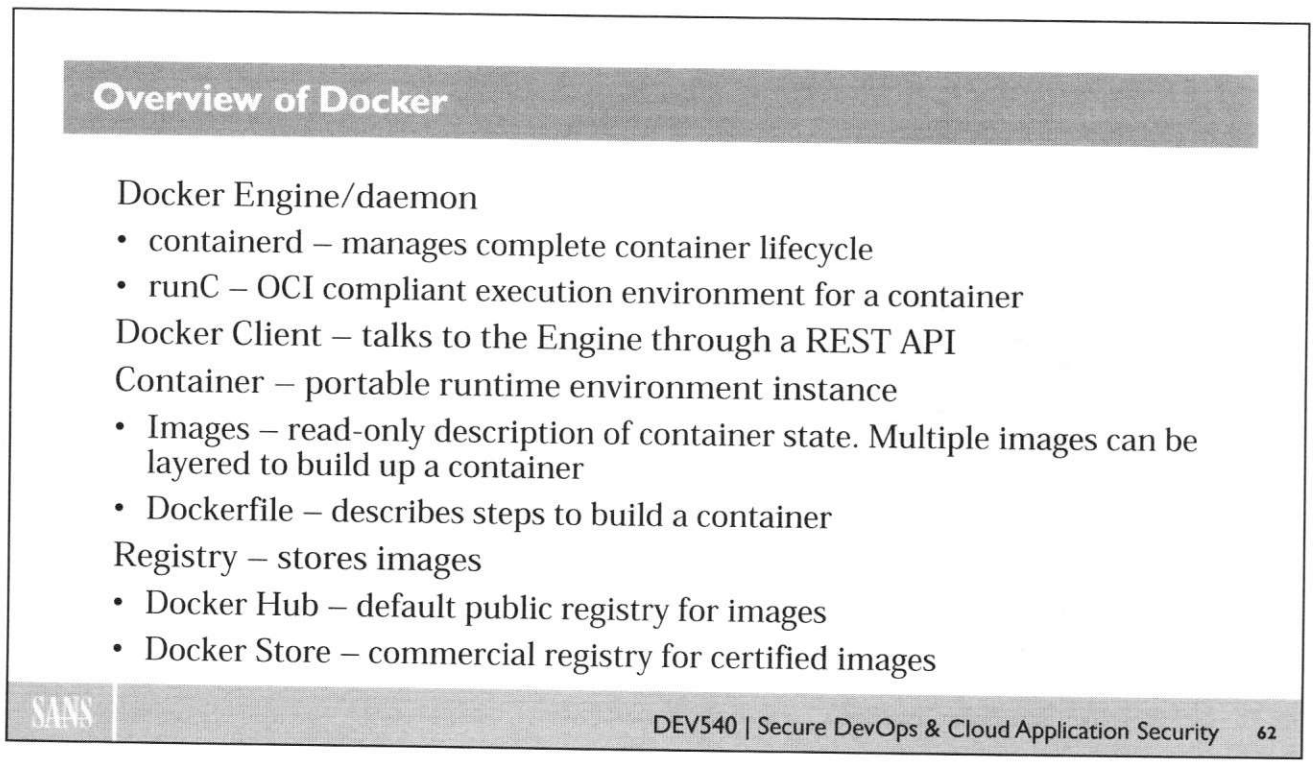
Containers do not emulate a hardware layer (there is no hypervisor). They use resource isolation features of the Linux kernel to isolate applications within a single instance of the OS:

- namespaces isolate the application's view of the operating environment, including process trees, network interfaces, user IDs and mounted file systems
- cgroups provide resource isolation, including CPU, memory, block I/O and network

See also: "StackOverflow: How is Docker different from a normal virtual machine?"
http://stackoverflow.com/questions/16047306/how-is-docker-different-from-a-normal-virtual-machine

image source: http://www.zdnet.com/article/vmware-buys-into-docker-containers/ (edited for clarity in print)

http://zdnet3.cbsistatic.com/hub/i/r/2014/10/04/a718b5bd-4bd3-11e4-b6a0-d4ae52e95e57/resize/770xauto/03d3ed9daa6c2b99b94c1cdf77a67737/docker-vm-container.png



Docker (https://www.docker.com/) is not just a lightweight virtualization and sandboxing technology. It is also a configuration management and deployment tool: a "shipping container for code". Docker makes it easy for developers to package up a complete application including runtime dependencies, by pulling down pre-defined images. They can then update their application code and test it directly, or ship the package out to test or production.

Docker is portable to local host, physical/virtual servers, or Cloud services on a range of platforms depending on the edition of Docker:

https://docs.docker.com/engine/installation/#supported-platforms

- Linux
- Cloud: Amazon EC2, Azure
- MacOS
- Windows 10 and Windows Server 2016 – as of Docker 1.12, available as native applications (July 2016)

Docker Engine and Client

The Docker Engine (https://www.docker.com/docker-engine) or daemon manages (builds and runs) containers on a system. It includes:

- containerd – the runtime environment for managing a container lifecycle
- runC (https://runc.io/) a lightweight runtime container which separates out the generic infrastructure plumbing in Docker, making Docker more modular, and allowing the plumbing functions to be re-used by other systems or services. It is an implementation of the Open Container Initiative (https://www.opencontainers.org/) a vendor-independent standard for defining runtime containers

The Docker Engine exposes a REST API to accept instructions from clients. The *docker* command talks to the Docker Engine daemon through this API.

Docker Images

Read-only, saved state of a container (the runtime environment). Images are layered on top of each other – first a base OS image, then runtime requirements, then application requirements and changes.

Docker images can be created interactively using a base image and the command *docker commit*

The command *docker history* shows how an image was built up. Tools like ImageLayers (https://imagelayers.io/) help visualize the layers in an image.

Dockerfile

A text file with commands in sequence to build a Docker image. Dockerfiles should be checked in to version control and managed like other source (including code reviews).

Docker Containers

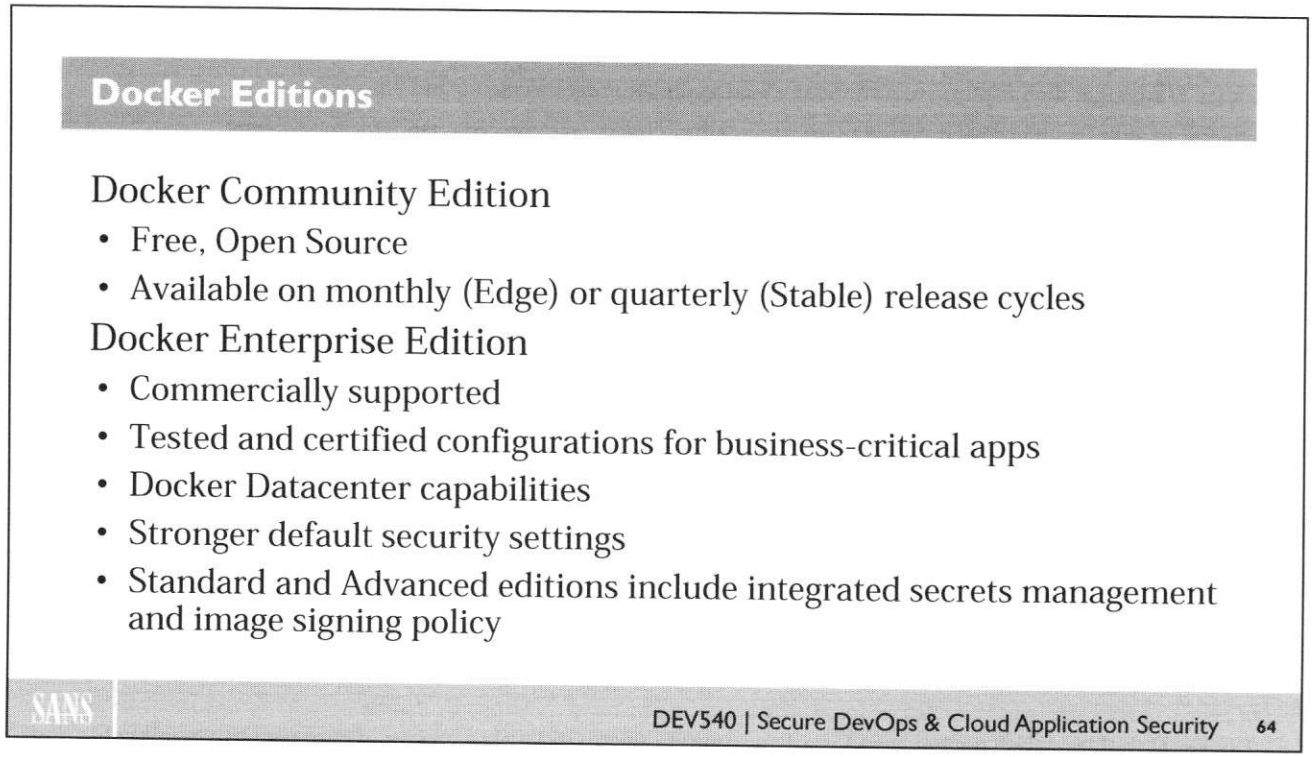
A runtime instance loaded from a Docker image. Each container can be run, started, stopped, and moved independently.

Docker Registries

Docker registries are public or private repositories used to upload or download images.

Docker Hub (https://hub.docker.com/) is a free, central registry of Docker images, where people can upload, download and share images. It includes community images, private images, and Official Repos which are managed by vendors. Official Repos (https://docs.docker.com/docker-hub/official_repos/) include standard based images for OSes such as Centos and Ubuntu, popular runtime stacks, etc. These repos are reviewed by Docker staff to ensure that they follow Docker best practices and are kept up to date by vendors.

Docker Hub is built on the Docker Registry project (https://docs.docker.com/registry/). You can use the Docker Registry to build your own internal repository of images that have been reviewed and approved. The Docker Trusted Registry (https://docs.docker.com/docker-trusted-registry/) is a commercially-supported, secure internal registry, with additional management functions.



Docker is available in different packaging options (all based on the same Open Source project)

Docker Community Edition (https://www.docker.com/community-edition)

- Free and Open Source, with maintenance from Docker for the latest shipping version
- Available in Edge (released monthly) and Stable (released quarterly) versions

Docker Enterprise Edition (https://www.docker.com/enterprise-edition)

- Commercial packaging and support for business-critical enterprise environments
- Runs certified containers and platform plugins from the Docker Store (https://store.docker.com/)
- Certified run-time OS and cloud environments: CentOS, Oracle Linux, RHEL, SLES, Ubuntu, Windows Server 2016, AWS and Azure
- Release quarterly (Stable) and supported for one year
- Available in different tiers: Basic, Standard and Advanced
- Standard/Advanced edition includes Docker Datacenter: multi-tenancy support with fine-grained RBAC and LDAP/AD, secrets management, image signing and image security scanning

Guidelines for Securing Docker EE:
https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Securing_Docker_EE_and_Security_Best_Practices

Working with Docker

Basic commands for working with images, containers, and repos

- docker run – Loads and runs container, downloads if needed
- docker ps – View running containers
- docker logs – Trace what's happening in container
- docker diff – What's happened on filesystem since container started
- docker history – Shows commands used to create image
- docker inspect – Shows internal details
- docker search – Searches repo
- docker stop – Halt the container; can be restarted or committed
- docker rm – Remove a stopped container from inventory

Commands to manipulate Docker images/containers

docker run {image]

Create and start a Docker container. If it can't find the image locally, it will download it from a registry automatically (first, it checks your local repo – if the image is not found, it will look for a public image in Docker Hub).

docker run --help

docker ps

View running containers

docker logs –f {container}

Trace what is happening in a container

docker diff {container}

What has changed on filesystem since the container started

docker history

Shows commands used to create an image

docker inspect {container/image}

Detailed internal information

docker search {image}

Searches repository

Or go to hub.docker.com and search online

docker stop

Halts a container; in this state, the container can be restarted, or committed to a new image

docker rm

Remove the container from inventory

Building Docker Images

You can build Docker images by hand

or...

Write repeatable images in Dockerfiles following standards/best practices

Dockerfile for Jenkins

```
FROM java:openjdk-8-jdk-alpine

RUN apk add --no-cache git openssh-client curl zip unzip bash ttf-dejavu

ENV JENKINS_HOME /var/jenkins_home
ENV JENKINS_SLAVE_AGENT_PORT 50000

ARG user=jenkins
ARG group=jenkins
ARG uid=1000
ARG gid=1000

# Jenkins is run with user `jenkins`, uid = 1000
# If you bind mount a volume from the host or a data container,
# ensure you use the same uid
RUN addgroup -g ${gid} ${group} \
    && adduser -h "$JENKINS_HOME" -u ${uid} -G ${group} -s /bin/bash -D ${user}

# Jenkins home directory is a volume, so configuration and build history
# can be persisted and survive image upgrades
VOLUME /var/jenkins_home
```


Building up a Docker image by hand

Start by selecting a base image for a Linux distro

docker pull {image:tag}

This will store the image on your host. By default, it will pull down the latest image in the repository

Load a container from this image with

docker run

Inside the running container, you can make whatever changes that you require, for example, adding a package, creating files...

After modifying the container

docker commit – m {what you did} –a {name of author} {container} {tag}

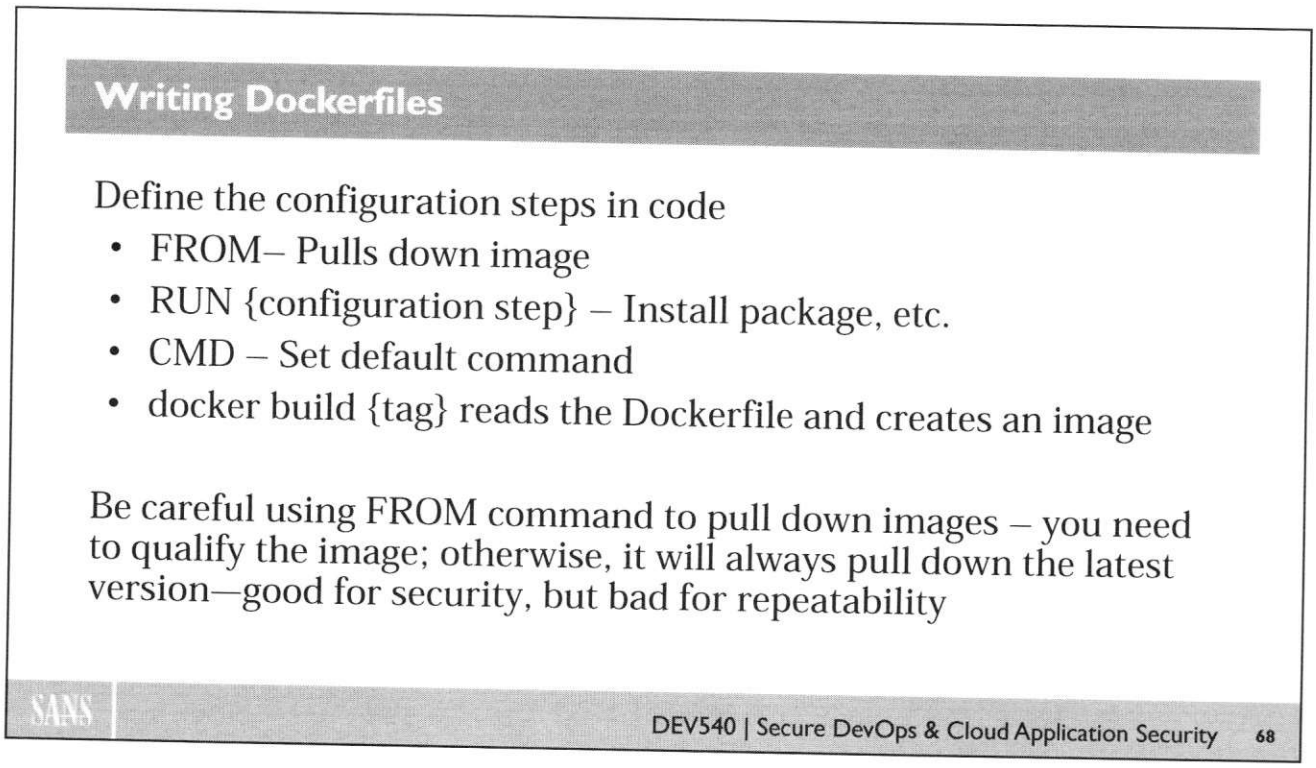
Creates a new image

Then you can push it up to a repository

docker login

docker tag {image} {namespace}

docker push {repository}



Writing a Dockerfile

It is easier to build Docker images with Dockerfiles – defining the configuration steps in code, and checking this code into version control. Define the base image and additional configuration steps (as layers on top of the base image):

FROM {base image:tag}

RUN {configuration step} – For example, install a package

EXPOSE {ports}

CMD {default command}

docker build {tag} – Reads dockerfile and creates image

Note that images created with dockerfiles are not necessarily repeatable: they may not create exactly the same image each time. This is because unless you use a specific tag in the FROM statement, docker will pull the latest image from a repository—the base image could have changed from the last time that you pulled it, unless you have full control over the image repositories involved. Even if you reference a specific tag for a build version, the image could have been patched.

Getting the latest version by default is good because it means you always have access to the latest features and fixes, but it can introduce breaking changes and make testing and debugging harder. If you need to pull exactly the same image, you need to pin it to a specific image digest, using

FROM image@digest

which will pull a specific image build. This means that you need to keep track of the digest id when you pull the original image.

docker images --digests=true

http://docs.docker.com/engine/reference/commandline/images/#listing-image-digests

https://github.com/docker/docker/pull/11109

Another issue with repeatability is that dockerfiles often include packages that are brought in from upstream (apt-get, yum, go get...) which may return new package contents, depending on whether the docker build cache is invalidated or not.

Using a digest is also useful if you want to make sure that you are pulling a specific patch, such as a fix for a security vulnerability.

Writing Dockerfiles – Best Practices

Docker has defined a set of best practices and patterns for writing Dockerfiles

Open source tools that will help you to create Dockerfiles that follow best practices:

Tool	Description
Rocker	Overcomes limitations with original Dockerfile design: Multiple FROM statements, new commands such as MOUNT, TAG, PUSH, and ATTACH.
Dockerize	Generates docker files using best practices for consistency and repeatability.

Best Practices for writing Dockerfiles

Docker and others have documented conventions and best practices for writing Dockerfiles:

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

http://crosbymichael.com/dockerfile-best-practices.html

http://jonathan.bergknoff.com/journal/building-good-docker-images

There are a couple of open source tools available that will help you to write standardized Dockerfiles and implement best practices:

Rocker: https://github.com/grammarly/rocker

Dockerize Me: https://github.com/fiunchinho/dockerize-me

Managing Docker Containers

From Docker (the organization)

- Compose – Define and run multi-container Docker apps
- Machine – Provision Docker containers
- Swarm – Clustering for Docker containers. As of 1.12, Docker Swarm Mode includes built-in TLS and automatic rotation of PKI certs for secure communications

From other sources

- cAdvisor – Container advisor, monitors running containers
- Kubernetes – To manage clusters of containers

Here are some basic, open source tools for managing Docker containers. Some of them are in early release stage and may not be stable.

Compose (https://docs.docker.com/compose/)
Tool for defining and running multi-container Docker applications

Machine (https://docs.docker.com/machine/)
Automatically create hosts, install Docker on them, and configure the Docker client to talk to them. You can use Machine to create Docker hosts on a local Mac or Windows box, on remote VMs, or on cloud platforms like AWS. Start, stop, inspect, or restart a host, upgrade the Docker client and daemon, and configure Docker clients to talk to a host.

Swarm (https://docs.docker.com/swarm/)
Native clustering for Docker: create and access a pool of Docker hosts, using standard Docker commands. Swarm Mode lets you create highly-available groups of Docker nodes which can pool resources so that you can manage multi-container, multi-host deployments. As of Docker 1.12, Swarm communications between masters and worker nodes is done through mutually-authenticated TLS out-of-the-box

cAdvisor (https://github.com/google/cadvisor)
Container Advisor monitors resource use and performance characteristics of running containers, resource isolation parameters, resource use and network stats. Developed by Google for its own containers.

Kubernetes (http://kubernetes.io/)
If you are trying to use containers at scale, you will need a centralized way to manage groups of containers. This is where container orchestration tools like Kubernetes come in. Kubernetes is based on Google's internal cluster-management system, Borg.
http://www.computerweekly.com/feature/Demystifying-Kubernetes-the-tool-to-manage-Google-scale-workloads-in-the-cloud

How Containers Fit into Continuous Delivery

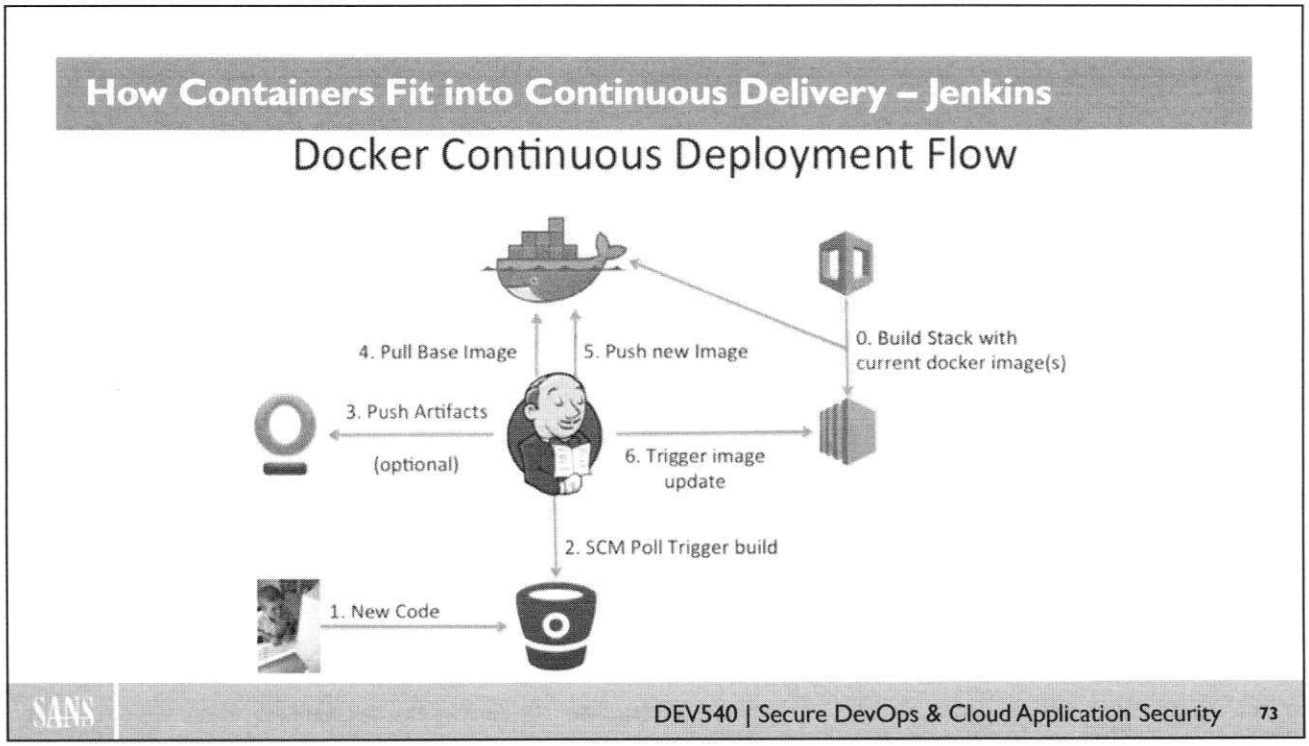
Containers can be built and shipped through CI/CD

- Pull image(s) from repo – or check out Dockerfile
- Build and package new image
- Push new image into repo
- Ship image to test stages

The steps for packaging and deploying applications using containers like Docker fit into Continuous Delivery. Docker commands and APIs and tools such as Docker Compose can be coordinated in a CI/CD pipeline to build and work with Docker images and repositories.

The abstraction level changes from the application binary to the container image which defines the application runtime environment with all of the dependencies included. This makes it easier and simpler to package and deploy changes: the entire runtime is moved through the pipeline stages. This brings the runtime definition to the front of the pipeline instead of leaving it to later. Run-time environment decisions must be made and tested early. You can also ensure that the runtime in production matches development and test – outside of any kernel differences in the runtime OS.

Roll-back is handled by deploying the previous version of the container.



Jenkins and Docker integration

There is good integration between Docker and Jenkins. The Docker team uses Jenkins and Docker to test Docker. The Jenkins team at CloudBees uses Docker and Puppet to build its own tools. This integration includes:

- Jenkins understands and uses Docker-based executors, and integrates with Docker image repos (to retrieve and store images)
- Workflow integration with Docker to orchestrate builds, tests, deployments
- Jenkins fingerprinting is extended for tracking Docker images across stages
- Docker images are first-class citizens in Jenkins operations
- Jenkins provides full traceability of all artifacts and actions involved

https://pages.cloudbees.com/rs/083-PKZ-512/images/Docker-Jenkins-Continuous-Delivery.pdf
https://www.docker.com/sites/default/files/UseCase/RA_CI%20with%20Docker_08.25.2015.pdf

There are some obvious use cases for Docker and Jenkins:

1. Install Jenkins using the official Docker image (https://hub.docker.com/_/jenkins/)
2. Use the Docker plugin for Jenkins to provision a Jenkins slave, run a build and tear it down (https://wiki.jenkins-ci.org/display/JENKINS/Docker+Plugin)
3. Execute Docker commands to create and manage Docker images within Jenkins using the Docker Build Step plugin (https://wiki.jenkins-ci.org/display/JENKINS/Docker+build+step+plugin)

The Jenkins Docker Commons Plugin (https://wiki.jenkins-ci.org/display/JENKINS/Docker+Commons+Plugin) provides APIs for using Docker from other plugins

Docker and Puppet/Chef

Puppet/Chef can be used to install and manage Docker and Docker containers

Docker can also be used to package and install Puppet or Chef...

The lab environment for this course uses Puppet to configure and manage Docker

Docker and tools like Puppet or Chef overlap with each other in a number of ways. They provide different approaches to packaging, configuring and installing packages and runtimes.

But they can also be made to play together in useful ways.

A popular Puppet module for Docker on Puppet Forge by Gareth Rushgrove provides Puppet functions for installing and configuring Docker, downloading images from the Docker Hub, automatically building images from Dockerfiles, and executing commands in containers:
https://forge.puppet.com/garethr/docker
https://github.com/garethr/garethr-docker
https://puppet.com/blog/simplify-managing-docker-puppet

In the lab environment for this course, Puppet is used to configure the Docker engine, download and build containers, and launch some of the containers used for core lab services (gitlab, DNS, Jenkins, etc.).

Similarly, Chef can be used to create container images, provision and configure a Docker host environment, launch container instances, and monitor and manage containers as they run.
https://www.chef.io/solutions/containers/

Ansible and Docker
http://www.ansible.com/docker
http://docs.ansible.com/ansible/docker_module.html

Salt and Docker
http://saltstack.com/saltstack-delivers-more-automation-docker-lxc-application-containers/
https://docs.saltstack.com/en/latest/ref/states/all/salt.states.dockerng.html#module-salt.states.dockerng

Containers in Production

- Docker – and containers in general – are being widely used in dev and test, but still ramping up for production workloads
- Google has proven that containers work in production at scale
- The Docker runtime environment is maturing, but is still a work in progress – Enterprise Edition is intended to address this
- Cloud-based consumer web and PaaS services are taking the lead
- Enterprises are running pilots and using Docker for greenfield projects (especially microservices)
- Financial organizations like Goldman and ADP are moving core production apps to Docker

Containers are not being used widely in production yet. One notable exception is Google, which uses its own containerization technology to run just about everything.
http://www.theregister.co.uk/2014/05/23/google_containerization_two_billion/

One shortcoming of using containers in production, especially at scale, is that most infrastructure management and monitoring tools work at the OS-level, and don't work with or look inside containers. This is especially a challenge with ephemeral containers which may only live for a few seconds at a time to service individual requests. A number of tools are emerging to address these challenges.

http://www.infoworld.com/article/2976930/application-virtualization/6-monitoring-tools-docker-containers.html

https://www.datadoghq.com/blog/the-docker-monitoring-problem/

In "Why Docker is Not Yet Succeeding Widely in Production", July 2015 (http://sirupsen.com/production-docker/), Simon Eskildsen lists some other factors that stand in the way of Docker being used in production. These include:

1. Image building – it is fast and easy to build simple images for small applications, but much more difficult to setup and manage images for large, complex applications
2. Garbage collection – it is also very easy to build up garbage in the form of old, obsolete images which need to be tracked and cleaned up
3. Speed of change – containers, especially Docker and rkt, are changing rapidly, which makes it difficult for operations to keep up. The rest of the runtime environment (monitoring tools, management tools like Machine and Swarm etc.) are immature
4. Run-time security – it is difficult to ensure that Docker is set up and used securely, especially in large environments. This is where tools like Twistlock, Aqua Security or StackRox come into play.
5. Dependence on new OS capabilities – Docker depends on new filesystem and kernel features that are not always stable and bug-free.

"Docker: Up & Running – Shipping Reliable Containers in Production", http://shop.oreilly.com/product/0636920036142.do

"Docker in production: Lessons from the Trenches" http://shop.oreilly.com/product/9781939902184.do

March 2016: Several enterprises are using Docker to roll out new greenfield projects (especially microservices). Other enterprises have projects underway to move core applications to Docker, including ADP and Goldman Sachs (which plans to move 90% of its applications to Docker).

"Docker, not production-ready? Not so, says Docker" http://www.infoworld.com/article/3046464/application-development/docker-not-production-ready-not-so-says-docker.html

For more on Goldman's Docker rollout: http://blogs.wsj.com/cio/2016/02/24/big-changes-in-goldmans-software-emerge-from-small-containers/

Docker Host OS Builds

Some OS builds are designed specifically to support Docker or other containers, especially in the cloud

- CoreOS
- RancherOS
- RedHat Atomic Host
- Snappy Ubuntu Core
- VMware Photon OS

They are stripped down, simple to update (can be patched on the fly) and are integrated with Docker

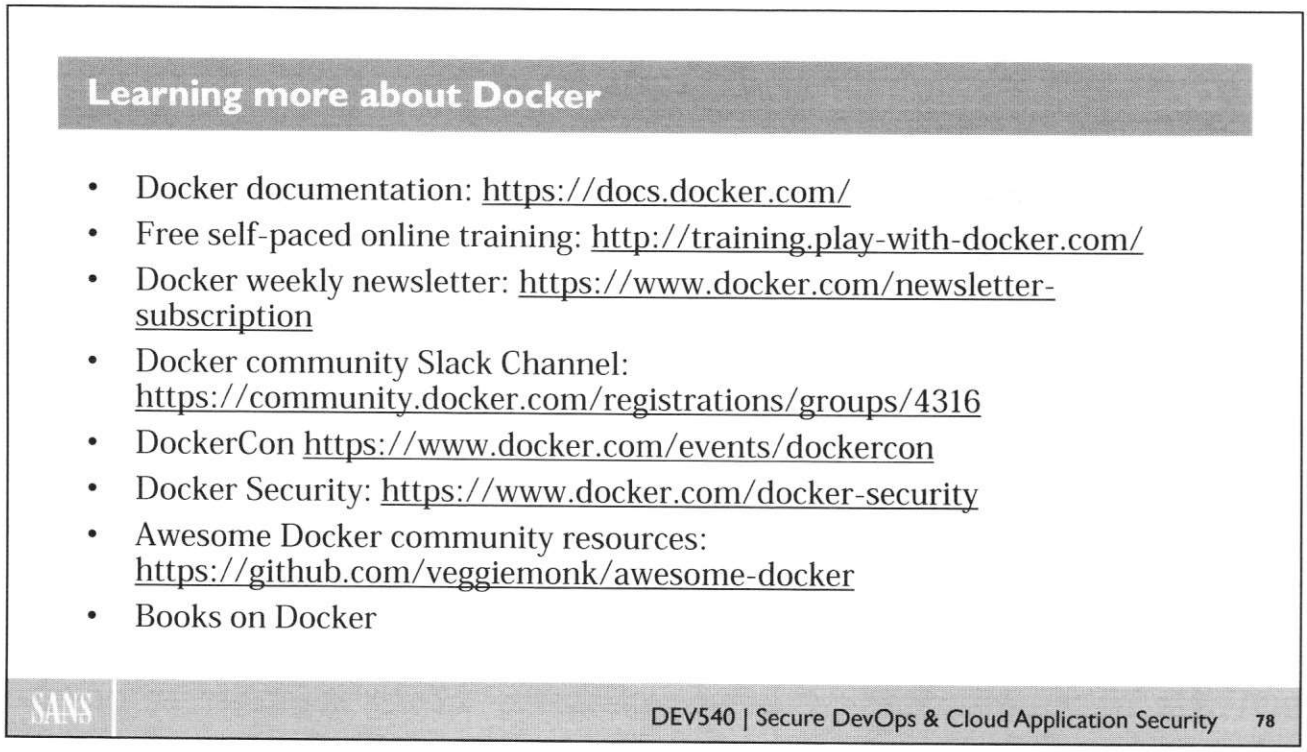
There is a number of stripped down host OS distributions specifically designed for running containers:

- CoreOS (https://coreos.com/using-coreos/)
- RancherOS
- RedHat Atomic Host (http://www.projectatomic.io/)
- Snappy Ubuntu Core (http://developer.ubuntu.com/en/snappy/)
- VMware Photon OS (https://vmware.github.io/photon/)

These OSes are lightweight and simple to update and are built to integrate with containers like Docker.

"Docker: A Comparison of Minimalistic Operating Systems" https://blog.inovex.de/docker-a-comparison-of-minimalistic-operating-systems

"Container OS Comparison" https://blog.codeship.com/container-os-comparison/



Learning about Docker

There are lots of books, articles, and posts on Docker – some of which are unfortunately out of date as Docker continues to change. Besides Docker's site itself, some good sources of information about Docker include:

Docker Up and Running: Shipping Reliable Containers in Production, by Karl Matthias and Sean P. Kane
http://shop.oreilly.com/product/0636920036142.do

Using Docker: Developing and Deploying Software with Containers, by Adrian Mouat
http://shop.oreilly.com/product/0636920035671.do

Docker Cookbook: Solutions and Examples for Building Distributed Applications, by Sébastien Goasguen
http://shop.oreilly.com/product/0636920036791.do

Awesome Docker on GitHub (https://github.com/veggiemonk/awesome-docker)
A comprehensive, community-built set of links on Docker and why it is awesome, popular tools, articles, books and resources

A cool Cheat Sheet for Docker (from Jan 2015)
http://container-solutions.com/docker-security-cheat-sheet/

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.

Container Security Advantages

Containers can offer some security advantages over default runtime

- Separation of applications on a server
- Only deploy what you need (limited dependencies and smaller attack surface)
- Makes it easier to setup and tear down instances – easier to patch
- Increases consistency – making hardening, auditing, and testing easier
- Control exactly what is in image with predefined instructions (which means you don't need to fiddle with setup in production)
- Sane OS defaults

Gartner (July 2016): Applications in containers (Docker) are more secure than applications on bare OS... provided that you set up and manage containers properly

Docker containers are transparent, reproducible runtime environments: the complete runtime environment (code, configuration) can be inspected, copied, deployed. You know exactly what is running and can set it up again as needed for testing. Docker provides additional protection over a bare metal runtime by isolating applications and by reducing the attack surface of the application – you only deploy the code and services that you need.

Docker also makes it easier and faster to push out patches. And Docker (like Puppet/Chef) increases consistency in configuration, which makes hardening, auditing, and testing easier.
"Docker: Introduction to Container Security", March 18, 2015
https://d3oypxn00j2a10.cloudfront.net/assets/img/Docker%20Security/WP_Intro_to_container_security_03.20.2015.pdf

Minimal distros
Docker allows you to build minimal/lean distros that only contain what you put into it. This means that you will have a smaller attack surface, and it is easier to patch – only these base containers need to be patched. However, many containers are built on base OS images which are quite fat. You may need to encourage developers to work from thin base images,

Sane Defaults

- Docker limits capabilities (around half are dropped)
- Applies AppArmor and SELinux policies by default
- Applies default seccomp filters for syscalls, filtering out over 50 system calls (as of Docker 1.10)

Gartner recommends using containers to increase runtime security
Increased isolation offered by containers and other advantages have led Gartner to take the position that "Applications deployed in containers are more secure than applications deployed on the bare OS" (July 2016)
http://blogs.gartner.com/joerg-fritsch/can-you-operationalize-docker-containers/
Gartner recommends deploying "Internet-exposed applications in Docker containers with best practice security". However, this depends on containers being properly and safely configured and managed; which is what we will look at over the next few slides

Container Security Issues

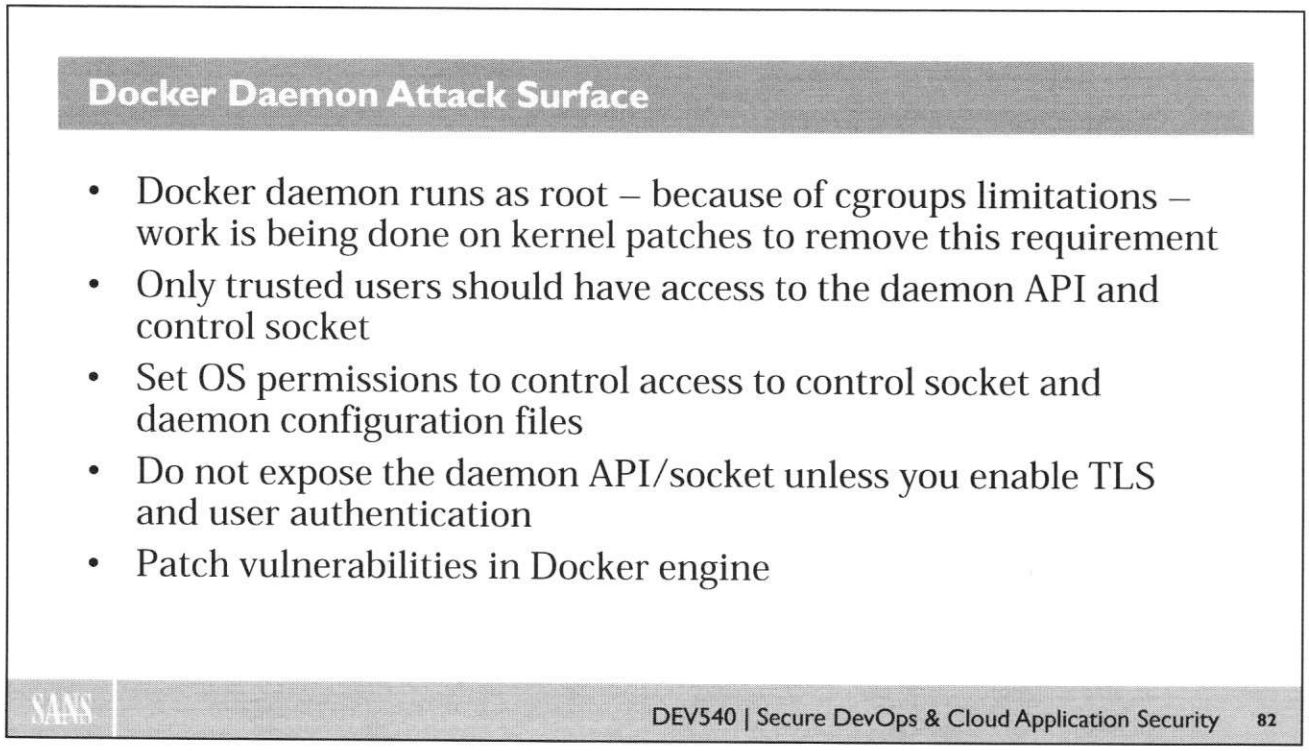
- Lightweight isolation: container separation is not as strong as VMs, container break out is a risk
- User namespace: process running as root in container has root in underlying host
  - User Namespacing added in Docker 1.10 (Feb 2016)
- Untrusted content – compromised and vulnerable images
- Container sprawl and limited visibility, especially at scale
- Docker Daemon presents its own attack surface

There are a number of security issues with containers (specifically Docker) that you need to manage. We will look at these issues in some detail over the next few slides.

A good summary of the current (Sept 2015) state of security for containers (especially Docker): "As containers take off, so do security concerns", by Maria Korolov, CSO Online, Sept 17, 2015 (http://www.csoonline.com/article/2984543/vulnerabilities/as-containers-take-off-so-do-security-concerns.html):

Docker 1.10 includes a feature called user namespacing which provides a mechanism to map the root user inside the container to a limited-privilege user outside the container. This feature is NOT enabled by default.

https://blog.docker.com/2016/02/docker-engine-1-10-security/



While using containers properly can reduce the system's attack surface, the Docker daemon itself offers an attack surface to bad guys. Anyone with access to the Docker daemon control socket or API effectively has root access to the container.

As an open source project, Docker is available for good guys and bad guys to inspect and learn how it works. This means that security researchers and other people in the community can help contribute to improving the security and robustness of Docker. But it also means that the code can be – and will be – inspected by bag guys for vulnerabilities that can be exploited. Docker does have security controls over community contributions: all contributions are reviewed, and Docker contracts outside experts for quarterly audits/pen tests of the code and infrastructure.

The Docker Daemon exposes an API to clients for Docker admin functions. This attack surface needs to be protected. Track and keep up with security patches to Docker itself:
http://www.cvedetails.com/product/28125/Docker-Docker.html?vendor_id=13534

Large parts of the Docker 1.11 CIS benchmark guide are focused on securing the Docker daemon:

- Section 1: Restricting access to docker group, auditing Docker daemon, and associated files
- Section 2: Docker daemon configuration considerations
- Section 3: Verifying that Docker daemon config files and socket permissions are set correctly

From the Docker site: (https://docs.docker.com/engine/articles/security/)

> Running containers (and applications) with Docker implies running the Docker daemon. This daemon currently requires root privileges, and you should, therefore, be aware of some important details.

First of all, only trusted users should be allowed to control your Docker daemon. This is a direct consequence of some powerful Docker features. Specifically, Docker allows you to share a directory between the Docker host and a guest container; and it allows you to do so without limiting the access rights of the container. This means that you can start a container where the /host directory will be the / directory on your host; and the container will be able to alter your host filesystem without any restriction. This is similar to how virtualization systems allow filesystem resource sharing. Nothing prevents you from sharing your root filesystem (or even your root block device) with a virtual machine.

This has a strong security implication: for example, if you instrument Docker from a web server to provision containers through an API, you should be even more careful than usual with parameter checking, to make sure that a malicious user cannot pass crafted parameters causing Docker to create arbitrary containers.

For this reason, the REST API endpoint (used by the Docker CLI to communicate with the Docker daemon) changed in Docker 0.5.2, and now uses a UNIX socket instead of a TCP socket bound on 127.0.0.1 (the latter being prone to cross-site-scripting attacks if you happen to run Docker directly on your local machine, outside of a VM). You can then use traditional UNIX permission checks to limit access to the control socket.

You can also expose the REST API over HTTP if you explicitly decide to do so. However, if you do that, being aware of the above-mentioned security implication, you should ensure that it will be reachable only from a trusted network or VPN; or protected (e.g. stunnel and client) SSL certificates. You can also secure them with HTTPS and certificates.

The daemon is also potentially vulnerable to other inputs, such as image loading from either disk with 'docker load', or from the network with 'docker pull'. This has been a focus of improvement in the community, especially for 'pull' security. While these overlap, it should be noted that 'docker load' is a mechanism for backup and restore and is not currently considered a secure mechanism for loading images. As of Docker 1.3.2, images are now extracted in a chrooted subprocess on Linux/Unix platforms, being the first step in a wider effort toward privilege separation.

Eventually, it is expected that the Docker daemon will run restricted privileges, delegating operations well-audited sub-processes, each with its own (very limited) scope of Linux capabilities, virtual network setup, filesystem management, etc. That is, most likely, pieces of the Docker engine itself will run inside of containers…

See also this post: http://security.stackexchange.com/questions/102323/risks-posed-by-docker-daemon-running-as-root

Understanding Docker Security Nathan Dec 19, 2015 – includes an update on authorization and authentication starting at 28:35

https://www.youtube.com/watch?v=w519CClzEuc

Docker Authentication and Authorization

Authentication to Docker Daemon

- Access to Linux domain socket
- Client certificate
- Improved authentication framework in development (Dec 2015)

Authorization

- By default, no authorization checking in Docker
- Authorization plugin framework implemented in Docker 1.10
- Authorization plugins available separately

"Understanding Docker Security" (https://www.youtube.com/watch?v=w519CClzEuc) Nathan McCauley, Dec 19, 2015 – includes an update on authorization and authentication starting at 28:35

Authentication

As of Docker 1.10, authentication controls for Docker daemon are weak. If you have socket access to the Docker daemon, you effectively have host root.

Work is underway on an authentication framework for Docker, to support LDAP/AD, 2FA, Kerberos, Unix named users, etc.

Open issue on implementing authentication controls in Docker daemon:
https://github.com/docker/docker/issues/13697

Authorization

By default, there is no authorization checking in the Docker daemon – anyone who has access to the Daemon has full control.

In Docker 1.10, an authorization plugin framework (https://github.com/docker/docker/pull/1536) has been implemented using technology from Twistlock, which allows you to implement your own access control rules.

After authentication, Docker requests (and results) are passed through an Authorization Plugin Framework, with a choice of plugins. The plugins act as interceptors that can and allow or deny a Docker API request based on granular access rules. These plugins are installed and configured will work the same as the current plugins for volumes and networking via the Docker plugin API.

Details on how to implement an Authorization/Access Control plugin can be found here:
https://github.com/docker/docker/blob/master/docs/extend/plugins_authorization.md

Handling Secrets in Docker

There is a lack of clarity around how to properly handle secrets in Docker, which means... that many people are doing this in an insecure way

Docker 1.13 (January 2017) includes initial, limited support for secrets when running in "swarm" mode. Build-time secrets are "on the way", as is support for using secrets with docker run https://blog.docker.com/2017/02/docker-secrets-management/

Best practice is to **use a secret keeper** like Vault

There are no clear guidelines or capabilities from Docker (yet) to handle secrets in a secure way. This means that people are forced to roll their own approach – and are making mistakes.

In "Secrets: write-up best practices, do's and don'ts, roadmap" (https://github.com/docker/docker/issues/13490) Sebastiaan van Stijn wrote up a list of "Features / hacks that are (mis)used for secrets" in Docker:

Environment Variables. Probably the most used, because it's part of the "12 factor app". Environment variables are discouraged because they are;

- Accessible by any process in the container, thus easily "leaked"
- Preserved in intermediate layers of an image, and visible in Docker inspect
- Shared with any container linked to the container

Build-time environment variables (#9176, #15182). The build-time environment variables were not designed to handle secrets. By lack of other options, people are planning to use them for this. To prevent giving the impression that they are suitable for secrets, it's been decided to deliberately not encrypt those variables in the process.

Mark .. Squash / Flatten layers. (#332, #12198, #4232, #9591). Squashing layers will remove the intermediate layers from the final image, however, secrets used in those intermediate layers will still end up in the build cache.

Volumes. IIRC some people were able to use the fact that volumes are re-created for each build step, allowing them to store secrets. I'm not sure this actually works, and can't find the reference to how that's done.

Manually building containers. Skip using a Dockerfile and manually build a container, committing the results to an image

Custom Hacks. For example, hosting secrets on a server, curl-ing the secrets and remove them afterward, all in a single layer. (also see https://github.com/dockito/vault)"

In another, related Docker issue (https://github.com/docker/docker/pull/9176#issuecomment-99542089), Diogo Monica of Docker's security team explains why storing secrets in environment variables is a particularly bad anti-pattern:

"env variables are the wrong way to pass secrets around. We shouldn't be trying to reinvent the wheel and provide a crippled security distribution mechanism right out of the box

When you store your secret keys in the environment, you are prone to accidentally expose them -- exactly what we want to avoid:

- Given that the environment is implicitly available to the process, it's incredibly hard, if not impossible, to track access and how the contents get exposed
- It is incredibly common having applications grabbing the whole environment—print it out, since it can be useful for debugging—or, even send it as part of an error report. So many secrets get leaked to Pagerduty that they have a well-greased internal process to scrub them from their infrastructure.
- Environment variables are passed down to child processes, which allows unintended access and breaks the principle of least privilege. Imagine that as part of your application you call to third-party tool to perform some action, all of a sudden that third-party tool has access to your environment, and god knows what it will do with it
- It is very common for applications that crash to store the environment variables in logfiles for later debugging. This means secrets in plaintext on disk.
- Putting secrets in env variables quickly turns into tribal knowledge. New engineers don't know they are there and are not aware they should be careful when handling environment variables (filtering them to sub-processes, etc.).
- Overall, secrets in env variables break the principle of least surprise, are a bad practice and will lead to the eventual leak of secrets."

He goes on to recommend using a secret keeper such as Keywhiz, Vault or Sneaker to handle secrets instead.

Benton Roberts describes a way to use secrets safely in Docker builds, using an open source tool called docker-ssh-exec (https://github.com/mdsol/docker-ssh-exec) which serves keys or other secrets over the private Docker-created network. http://techblog.mdsol.com//2015/10/30/Using-Secrets-Safely-in-Docker-Builds.html

See "Understanding Docker Security" Nathan McCauley, Director of Security, Docker, Dec 19, 2015 (discussion of secrets management starts at 33:37) https://www.youtube.com/watch?v=w519CClzEuc

Container Security – Isolation

- Containers add a layer of protection
- However, container isolation is not complete
  - User namespacing is relatively new (v1.10)
  - Vulnerabilities in Docker Engine can break isolation
- Isolate containers in a VM or on separate physical boxes for multi-tenant apps
- Don't mix tiers on the same physical box—public network-facing apps should be on different boxes

Isolation in containers is principally implemented through namespaces, which provide a separate view of different system resources (IPC, network, mount, PID, etc). For example, you only see, and can only work with, the processes inside your container. But it is important to understand that container isolation is not as strong as VM isolation. You should assume that it is possible to break out of a container.

There have been a number of serious vulnerabilities in Docker for example which allow users to breach container isolation, including:

- [CVE-2015-3627] Insecure opening of file-descriptor 1 leading to privilege escalation
- [CVE-2015-3629] Symlink traversal on container respawn allows local users to escape containerization
- [CVE-2015-3630] Read/write proc paths allow local users to modify the host, obtain sensitive information and perform protocol downgrade attacks

If you are concerned about weaknesses in isolation and container-breakout, you need to:

- Make sure that you don't mix tiers on the same physical box: public network-facing apps should be physically isolated from application worker services and database services
- Isolate containers in separate VMs or on separate physical hosts in a multi-tenant environment. Some firms actually choose to implement only 1 container per VM, to provide two layers of containment and isolation. This improves security at the cost of hardware.

VMware has done a series of benchmarks which show that when properly tuned, there is only a small percentage performance hit from running containers inside a vSphere VM over bare metal Linux. https://blogs.vmware.com/performance/tag/docker

https://blog.docker.com/2016/02/docker-engine-1-10-security/

Container Security – Isolation - Namespacing

- The Docker community has been working on user namespacing for a long time
- User namespacing for root supported in Docker 1.10: root inside the container is not root outside of the container
- **NOT enabled by default**: --userns-remap=[USERNAME]
- More work is being done to improve isolation
- Avoid running applications in containers as root unless you have no choice

Docker implements 5 namespaces to work with Linux:
1. process
2. network
3. mount
4. hostname
5. shared memory

Other important namespaces are outside of the container, including all devices and file systems under /sys, and user namespaces. If a user or app has root within the container, they could compromise the underlying OS.

Users are not fully namespaced in Docker. User namespacing for root is implemented in Docker 1.10. In this phase, the root user is remapped so that root inside Docker will have access to privileged operations inside the container, but not outside. This capability **is not enabled by default**.
https://docs.docker.com/engine/reference/commandline/dockerd/#/daemon-user-namespace-options

For an online demo of how this feature works: https://asciinema.org/a/5uyrknsjg7u2fad6ii0wgizd4?speed=2
For a detailed discussion, see "Rooting out Root: User Namespaces in Docker", by Phil Estes (the primary implementer of the feature)
http://events.linuxfoundation.org/sites/events/files/slides/User%20Namespaces%20-%20ContainerCon%202015%20-%2016-9-final_0.pdf
Additional work is being/may be done to improve user namespace isolation. As of November 2016, it is waiting on kernel changes. See https://github.com/docker/docker/issues/15187 and
http://www.slideshare.net/PhilEstes/how-secure-is-your-container-containercon-berlin-2016

Until user namespacing is fully implemented and proven to work, you should avoid running applications in a container as root – if an attacker is able to break out of the container, they will have root privileges at the OS level, which means that they can compromise other containers. Always try to run applications as a non-privileged user (through the Docker USER statement). And take extra steps to harden the container to prevent root break-out.

Container Security – Image Poisoning

- Images from public registries may contain vulnerabilities – or malware
- Mitigate this risk by vetting images from public repositories
  - Official Repos
  - Image Scanning
- Docker Trusted Registry
- Add manual reviews/scanning on images after downloading and before checking in/pushing to repo

Docker makes it easy for developers to download and work with pre-built images. This also makes it easy for them to introduce vulnerabilities by accident. A study by BanyanOps May 29, 2015 (http://www.infoq.com/news/2015/05/Docker-Image-Vulnerabilities) found that "over 30% of Official Images in Docker Hub contain high priority security vulnerabilities".

The risks of using pre-built Docker images are similar to the risks of developers using open source software libraries and other components:

- Downloading out-of-date components which contain known security vulnerabilities, and inheriting those vulnerabilities in your system
- Downloading compromised software that contains malware or that has been otherwise tampered with
- Vulnerabilities in base images can cascade quickly, as additional images are quickly built on these bases and distributed and shared.

While Docker is taking active steps to improve the security of images hosted in Docker Hub (especially for Official Repositories), you need to understand and manage the risks of building on public images. You must vet images from repositories to make sure that you aren't accidentally installing a Trojan Horse or a vulnerable runtime. This includes:

- Verifying the provenance of images (the chain of trust)
- Ensuring that the image has been scanned/reviewed for vulnerabilities
- Testing and reviewing images before allowing them to be checked in to your own registry – this is a natural gate point for adding security checks

Container Security – Image Poisoning – Official Repos

- Official Repos are curated images on Docker Hub
- Docker and vendors work together to review and check the images in these repos
- Official repos are scanned for vulnerabilities using Docker Security Scanning

Docker Hub Official Repositories (https://docs.docker.com/docker-hub/official_repos/)

Official Repositories are a curated set of images on Docker Hub, including base OSS images and popular runtime platforms and data stores. A team at Docker works with upstream software maintainers, security experts, and the broader Docker community to ensure the quality and security of images in official repos. These images are reviewed to ensure that they follow good practices and provide clear documentation, and are regularly checked that they are up-to-date and free of vulnerabilities.

Docker strongly recommends that you use official repos where possible.

Container Security – Image Scanning

- Docker Security Scanning (Nautilus)
- Scans official repos and available for Docker Cloud private repos as an add-on service
- Scans for known vulnerabilities in dependencies

Docker Security Scanning (Project Nautilus)

As of November 2015, Docker started to automatically scan official repository images hosted on the Docker Hub for security vulnerabilities (Project Nautilus). "Moving forward, the plan is to make Nautilus a self-service capability that anyone can run for application images."

In May 2016, Docker announced general availability of Docker Security Scanning, as an add-on to Docker Cloud private repositories and for Official Repositories on Docker Hub.
https://blog.docker.com/2016/05/docker-security-scanning/

Docker's image scanning includes scanning at the code level and comparisons against CVEs, as well as other [currently] undisclosed deep content analysis and vulnerability scanning techniques, including the use of third-party providers. The scanner walks the file systems of images, identifies code/packages to build up a bill of materials (BOM), and maps this against databases of known vulnerabilities. It also automatically provides email notifications when new vulnerabilities are reported to a CVE database.

Container Security – Scanning Tools

Open Source	Commercial
CoreOS Clair	Docker Security Scanning (fka Nautilus)
OpenSCAP plugin	Twistlock
Banyan Ops Collector	Aqua Container Security Platform
Dockscan	Flawcheck – acquired by Tenable
Drydock	IBM Docker Security Vulnerability Advisor
Batten	Black Duck Hub
Lynis Docker plugin	Tenable Nessus (6.6+)
Docker Bench and Docker Actuary	Sonatype Lifecycle Container Analysis

In addition to Docker's own scanning of official repo images in Docker Hub, there are several tools available to run different types of security scans of containers, especially Docker containers and registries:

Docker Bench (https://github.com/docker/docker-bench-security)

Scans containers against the Docker CIS benchmark

Docker Actuary (https://github.com/diogomonica/actuary)

Follow-on to Docker Bench, with customizable profiles.

CoreOS Clair (https://github.com/coreos/clair)

Statically scans containers (Docker and CoreOS rkt), identifies packages or components in the image layers and highlights known vulnerabilities

https://coreos.com/blog/vulnerability-analysis-for-containers/

http://www.infoq.com/news/2015/12/clair-docker-vulnerabilities

Twistlock (https://www.twistlock.com/product/vulnerabilitymanagement/)

A commercial solution which scans container images in registries and in development and production runtime environments. Detects and reports vulnerabilities in the Linux distribution layer, application frameworks, and custom application packages. Can be integrated into CI.

Twistlock provides a limited, unsupported free Developer Edition (limited to 10 repos and 2 production hosts). The free version only includes open source threat feeds.

Aqua Security Peekr (https://www.aquasec.com/products/aqua-peekr/)

Peekr is a SaaS scanning solution for Docker from Aqua Security, fka Scalock (a commercial container security/management provider). Peekr scans images against known vulnerability signatures from public open source vulnerability databases and through a partnership with WhiteSource. It scans images in Docker Hub, Amazon ECR, CoreOS Quay, and V1 and V2 Docker Private Registries. Peekr also runs images in sandboxes to detect suspicious or malicious behavior ("Active Container Inspection").

As of Feb 2016, Peekr is free on a limited basis: "Currently, new Peekr users can run up to ten scans per month. However, this quota is increased by inviting others to use Peekr. Every time someone accepts a Peekr invitation, the quota of both the inviter and the invitee grow by five scans."

http://blog.aquasec.com/introducing-peekr-scalocks-free-security-scanner-for-container-images

OpenSCAP plug-in for scanning Docker and CoreOS rkt containers (https://github.com/OpenSCAP/container-compliance)

http://www.open-scap.org/resources/documentation/security-compliance-of-rhel7-docker-containers/

http://cdn.oreillystatic.com/en/assets/1/event/129/Using%20open%20source%20tools%20to%20secure%20containers%20and%20clouds%20Presentation.ppt

IBM Docker Vulnerability Advisor: built into the IBM Bluemix cloud platform

https://developer.ibm.com/bluemix/2015/07/02/vulnerability-advisor/

Lynis (https://cisofy.com/lynis/)

Open source security auditing tool includes auditing for Docker as an enterprise plugin
https://cisofy.com/lynis/plugins/docker-containers

Black Duck Hub (https://www.blackducksoftware.com/security/open-source-security/software-container-security)

Commercial solution which scans and identifies open source software throughout code base, including scanning Docker containers, cataloging all images and open source software inside the images, and tracking vulnerabilities in this software

FlawCheck (https://www.flawcheck.com/features/container-inspection/)

Offers cloud-based or on-premise vulnerability scanning of private registries. Acquired by Tenable in October 2016

Collector (https://github.com/banyanops/collector)

Collector, from Banyan Ops, is an open source framework for static analysis of Docker images
http://securedocker.org/

Dockscan (https://github.com/kost/dockscan)

Early-stage vulnerability assessment and audit tool for Docker containers

Drydock (https://github.com/zuBux/drydock)

Flexible open source security assessment tool, allows you to create and use custom audit profiles – currently checks only against CIS Docker 1.6 Benchmark

Batten (https://github.com/dockersecuritytools/batten)

Open source hardening and auditing tool for Docker hosts and containers

Nessus (https://www.tenable.com/blog/auditing-docker-with-nessus-66)

In 6.6, Nessus adds support for auditing and scanning Docker hosts and containers.

- The Docker Service Detection plugin detects Docker installs and enumerates active Docker containers
- Nessus can scan the underlying host for vulnerabilities
- Nessus checks the host and containers against the Docker CIS 1.6 benchmark.

In Jan 2017, Sonatype announced a Lifecycle Container Analysis solution (https://www.sonatype.com/containers) as part of its Nexus suite of dependency management tools

For a good overview of Docker security auditing and vulnerability assessment tools (as of Dec 2015), see this post

http://blyx.com/2015/12/01/docker-security-tools-audit-and-vulnerability-assessment/

Docker Content Trust

- Allows you to digitally sign tagged images before pushing to the Docker Hub
- Ensures that the tagged image was signed when pulling
- Based on The Update Framework (TUF) open source framework for software update systems
- Yubikey integration

Docker Content Trust is an opt-in feature that uses digital signatures and a trusted update protocol to verify the publisher of Docker images, and to ensure that these images have not been tampered with. Using Docker Content Trust, you can ensure that you are using the correct images provided by a publisher. It is based on The Update Framework, an open source specification for software update systems: http://theupdateframework.com/

Before a publisher pushes an image to a remote registry, Docker Engine signs the image locally with the publisher's private key. When you later pull this image, Docker Engine uses the publisher's public key to verify that the image you are about to run is exactly what the publisher created, has not been tampered with and is up to date.

As of 1.8, Content Trust is only supported in the Docker Hub. Experimental integration with Docker Trusted Registries was added in 1.9.

Content Trust changes the way that you create an image and pushes it to the Docker Hub, and what images you can pull from a registry. Enabling Content Trust forces client-side signing and verification of image tags. Images are digitally signed and used to verify the integrity of the image and the publisher id. Content Trust is associated with an image tag – it is possible to create images that are signed and unsigned, depending on the tag.

Enabling Content Trust forces a client to use only signed image tags when pulling an image from the Docker Hub (unless you are pulling based on an explicitly content hash value, or unless you use the –disable-content-trust command flag).

export DOCKER_CONTENT_TRUST=1

The first time that you push a tagged image with Content Trust enabled, you will be asked to create a new root signing key and a repository key, and corresponding passphrases for these keys. Docker recommends that the passphrase for the root key and the content key-pair should be randomly generated and stored in a password manager. On subsequent pushes of a tagged image, you will be asked to supply the root key passphrase and enter a new repository key passphrase.

Yubikey hardware devices can be used to hold the root key and generate a per-repository key for digitally signing Docker images for a particular registry.

Docker Trusted Registry

Extra cost, on-premises or private cloud registry for enterprises

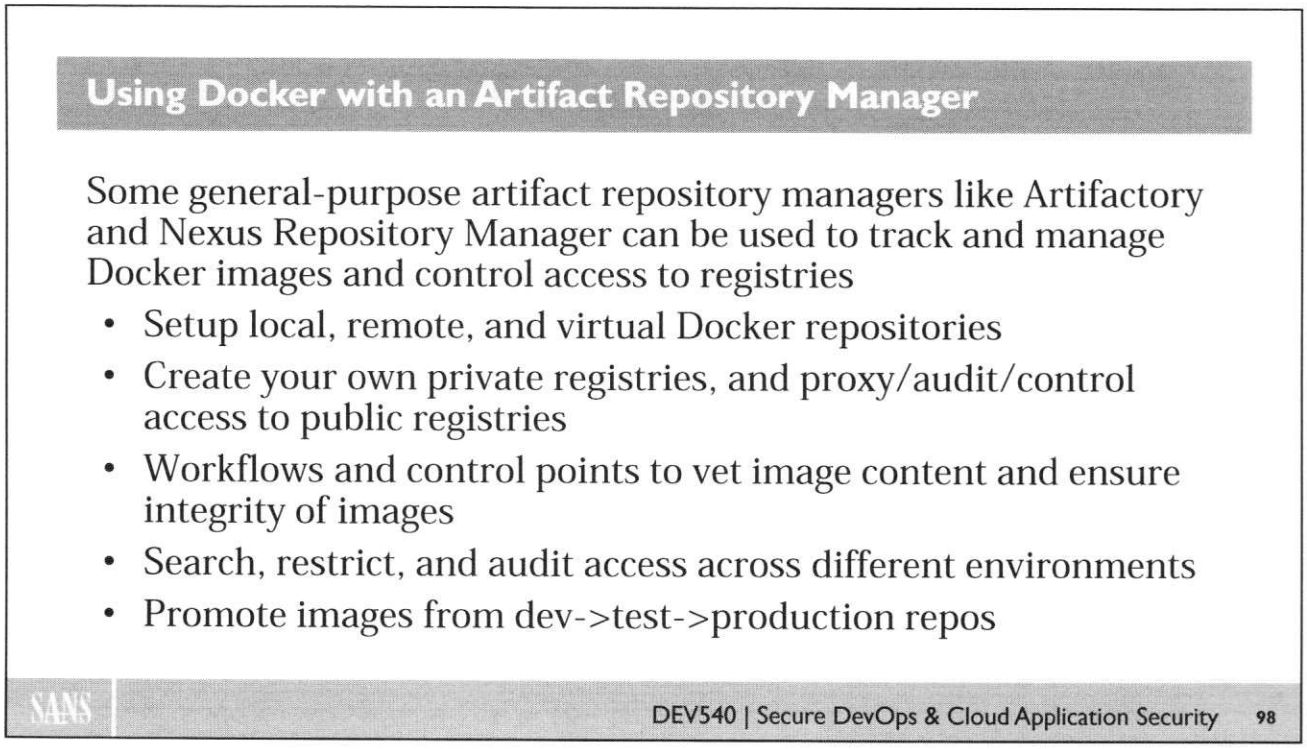
- Use instead of Docker Hub to manage your own images and approved third party images
- Built-in authentication and fine-grained access control

Docker Trusted Registry

Docker Trusted Registry (https://www.docker.com/docker-trusted-registry) is a for-pay, on-premises/private cloud registry that organizations (especially organizations in regulated industries) can manage for themselves. It is designed to be used instead of public registries like Docker Hub. The Trusted Registry integrates into Active Directory/LDAP for authentication and supports Role-Based Access Control for permissioning.

Support for Docker Content Trust is being added to the Docker Trusted Registry (experimental support available in Docker 1.9).

See case study from Booz Allen Hamilton: "Secure DevOps with Docker Trusted Registry": https://youtu.be/slsAzknseTg



General-purpose artifact repository managers like Artifactory and Sonatype's Nexus Repository Manager can be used to manage Docker images and repositories as well as other build and deployment artifacts.

Artifactory and Docker

Artifactory (https://www.jfrog.com/artifactory/) is a general-purpose artifact repository manager, which supports multiple local, remote and virtual artifact repositories. You can use Artifactory to set up your own secure, private Docker repositories. It provides secure and audited, transparent access through the Docker Registry API to

- Local Docker repositories
- Remote Docker repositories (e.g. Docker Hub) through a caching proxy
- Virtual repositories which are wrappers over aggregated local and/or remote repos

Artifactory provides a centralized, common interface to multiple repositories. Engineers can push images to, or pull images from, Artifactory's Docker repos in the same way that they would work with a Docker registry.

Artifactory has built-in workflows for promoting artifacts. This can be used to move Docker images through different acceptance and testing stages – from a dev repo to test, staging and production, without using tags/labels. This makes it possible to track which containers are deployed in which environments.

https://www.jfrog.com/blog/docker-registry-to-production/

Nexus Repository Manager and Docker

As of version 3.0, Sonatype's Nexus Repository Manager can also be used to manage Docker containers. For more information see: https://books.sonatype.com/nexus-book/3.0/reference/docker.html

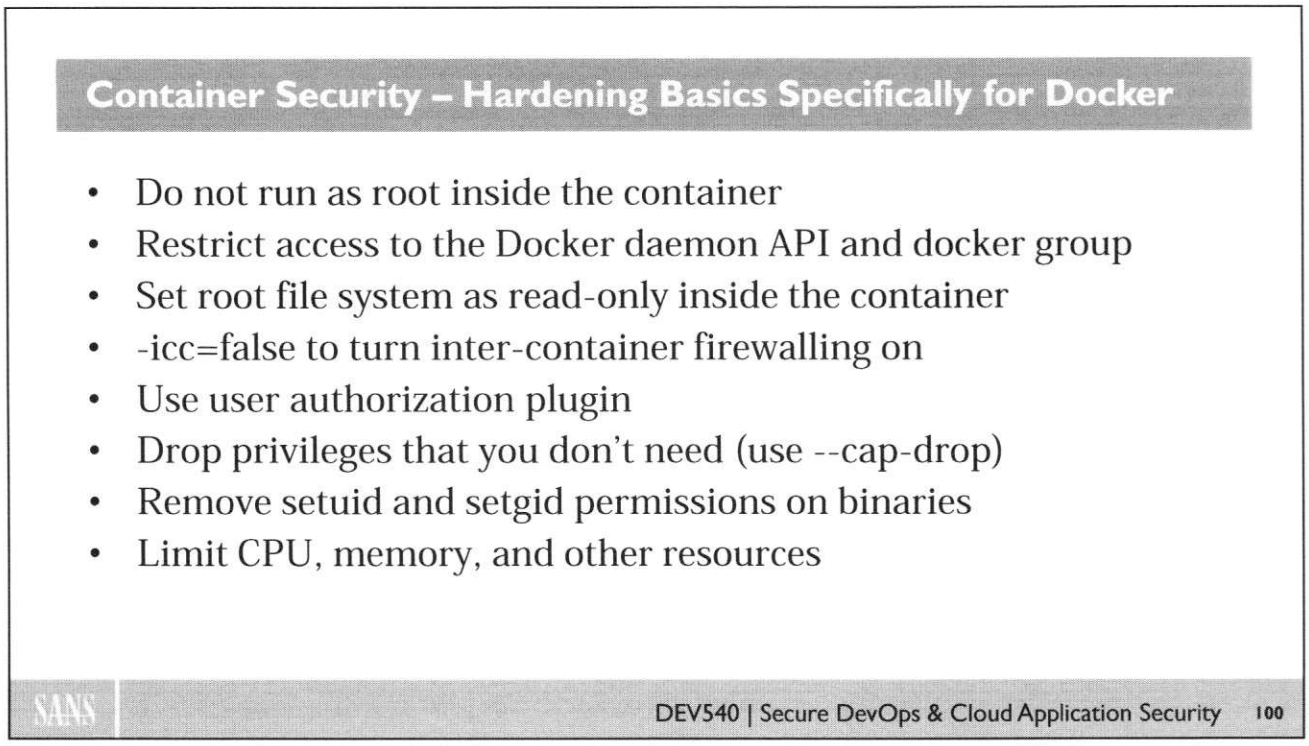
Container Security – Hardening Basics for All Containers

- Harden the underlying host OS and kernel
  - Keep OS especially kernel patches up to date
  - Consider using grsecurity/PaX, SELinux or AppArmor for Linux hosts (see later slide)
  - Follow standard host hardening guidelines
- Keep containers up to date
- Use a secrets manager for keys, passwords and other credentials
- Isolate apps or customers in separate VMs
- Only install what you need in the image – minimize the attack surface

Containers are not secure by default. This includes Docker. Although Docker has continued to tighten up security, the configuration defaults for Docker containers are biased to needs and priorities of developers. You will need to harden the default container configuration and the runtime environment. Understand that hardening can negate some of the ease-of-use and speed-to-provision advantages of using containers, by adding more overhead and steps, increasing friction and adding delays.

Some basic guidelines apply to securing any kind of container:

1. Keep the underlying host OS up to date – especially the kernel. Track kernel-level security vulnerabilities and apply patches.
2. Harden the host configuration. Follow standard OS security recommendations, and consider using a stripped down, minimalistic base OS.
3. Keep the container up to date – track container security vulnerabilities and apply patches. For Docker CVE details, see: http://www.cvedetails.com/vendor/13534/Docker.html
4. Use a secrets manager to store and manage passwords, keys and other credentials
5. Isolate apps or different customers in separate VMs to prevent the possibility of container breakout
6. Only install what you need in the image, to minimize the attack surface. Use minimal base images.



Some basic hardening steps for Docker:

- Do not run as root inside the container. In the base image, RUN useradd, add a USER directive in Dockerfiles and start the container with –u. If this is not possible, enable user namespace remapping (as of Docker 1.10).
- Restrict access to the Docker daemon (API and control socket), and to the "docker" group – by default, anyone with access to the Docker command API has full admin privileges (root) within the container
- Set the root file system as read-only inside the container. This prevents the container from being modified or tampered with at runtime. Define a separate container volume for writing information. See Docker 1.13 CIS benchmark recommendation 5.12
- Run with –icc=false to turn inter-container network firewalling on. This disables inter-container communication, preventing cross-host attacks and cross-container attacks within the same host.
- Implement user authorization using the authorization plugin introduced in Docker 1.10 to limit access to the Docker Engine
- Drop privileges that you do not need, as quickly as possible (use --cap-drop)
- Remove setuid and setgid permissions on binaries (or remove the binaries)
- Limit CPU (-c), memory (-m), restart attempts (-on-failure policy) and user limits (ulimits)

Container Security – Hardening the OS Kernel

Docker relies on the security of the OS kernel. Take extra steps to harden the kernel

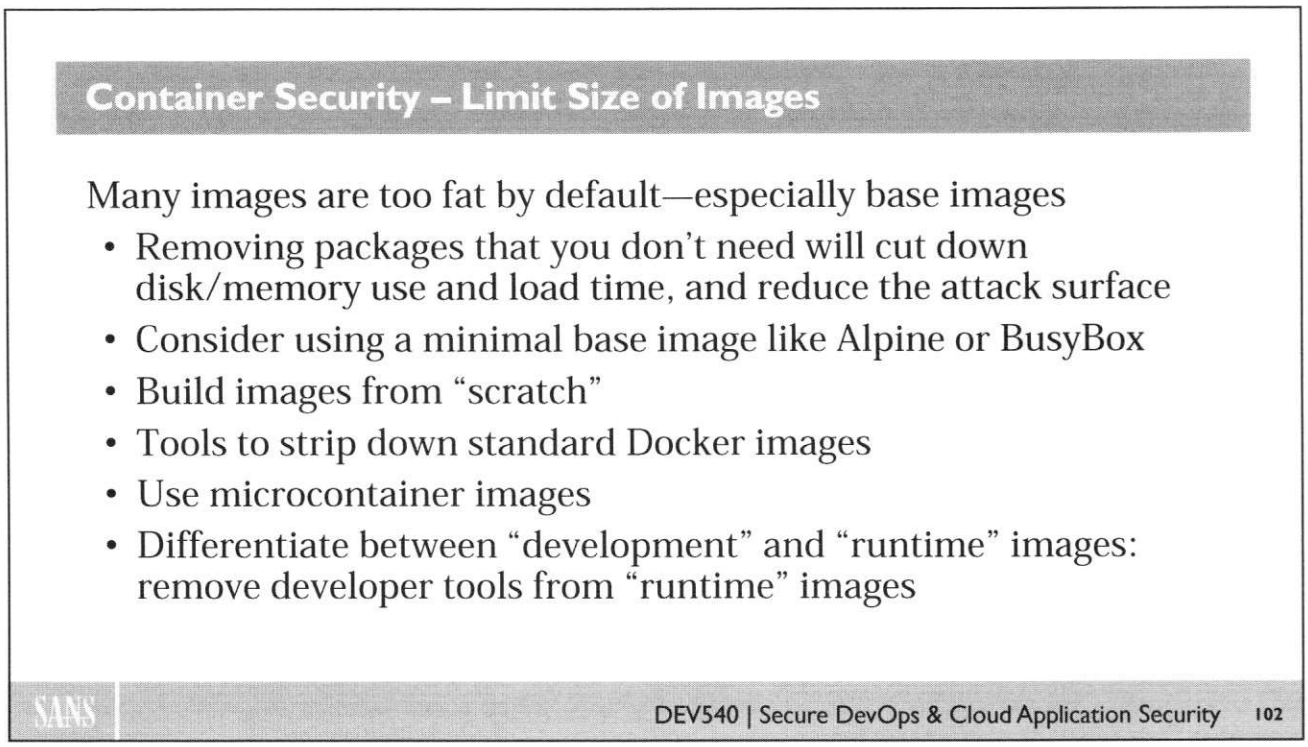
- grsecurity and PaX – l0w-level kernel security patches
- **SELinux**
- AppArmor – See project Bane by @jfrazelle

Hardening the OS/Kernel

Docker strongly recommends that you take advantage of Linux security extensions in production environments, such as:

- grsecurity and PaX (https://grsecurity.net/): a set of security patches that provide kernel-level safety checks at compile time and runtime.
- SELinux (https://selinuxproject.org/page/Main_Page) is a Mandatory Access Control (MAC) system which allows you to (and obliges you to) set fine-grained access control rules. It is available on RHEL and Centos, Fedora, Debian, Ubuntu and openSUSE distros. SELinux is famously difficult to set up correctly.
- AppArmor (http://wiki.apparmor.net/index.php/Main_Page) is a Mandatory Access Control (MAC) system which works by applying profiles to different processes. It is available on Debian, Ubuntu and openSUSE Linux distros. Docker automatically applies a default AppArmor profile to each launched container, which provides some runtime protection policies (https://docs.docker.com/engine/security/apparmor/). If you want to set up your own custom AppArmor policies, check out the Bane project by @jfrazelle: https://github.com/jfrazelle/bane

The lab VM uses SELinux in enforcing mode.



Many standard images – especially base OS images – are fat by default. They contain packages that you don't need and won't use. Cutting these packages out will:

- Reduce the disk and memory footprint of the container
- Shorten container load time
- Reduce the attack surface of the runtime container

Cutting down the size of your base image is listed as recommendation 4.3 in the CIS Docker 1.11 Docker Benchmark guide.

Use the following commands to audit:

docker ps –quiet {list all running instances of containers on a host}

docker exec $INSTANCE_ID rpm –qa {for each instance, list the packages installed on the container}

Minimal Base Images

Standard base images will include packages that you don't need and won't use. Some minimal base images for Linux distros include:

- Alpine (https://hub.docker.com/_/alpine/)
- BusyBox (https://hub.docker.com/_/busybox/)

Building images from Scratch

Docker provides an empty default image called "scratch" (https://hub.docker.com/_/scratch/) which can be used to build up base images or other minimalist images.

Tools to strip down standard images

There are some free tools available which can be used to strip down standard images (not just OS base images):

- Strip Docker Image (https://github.com/mvanholsteijn/strip-docker-image)
- Docker Slim (https://github.com/cloudimmunity/docker-slim)

"How to create the smallest possible Docker container of any image" http://blog.xebia.com/how-to-create-the-smallest-possible-docker-container-of-any-image/

"Creating Smaller Docker Images" https://www.ianlewis.org/en/creating-smaller-docker-images

Microcontainer Images

Engineers at Iron.io have created a set of "microcontainer" images (https://github.com/iron-io/dockers): stripped down images for major programming language, which contain only the OS libraries and language dependences required to run an application. These images are all built up from the Docker "scratch" image.

"Microcontainers – Tiny, Portable Docker Containers" https://www.iron.io/microcontainers-tiny-portable-containers/

Development Containers vs Run-time Containers

Containers built by developers may include developer tools and other dependencies that are not required at runtime and should be removed.

Docker Security – Dropping Capabilities

- By default, Docker limits capabilities
- You can improve security by dropping additional capabilities that are not needed, or that may be dangerous
  - Run-time argument --cap-add or --cap-drop
- --privileged argument allows a Docker container to access all devices on the host and override AppArmor/SELinux restrictions

In order to reduce the attack surface in a container, Docker allows you to only allow the specific capabilities that processes in the container require to run.

From the Docker site (https://docs.docker.com/engine/articles/security/):

> By default, Docker starts containers with a restricted set of capabilities. What does that mean? Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system. Processes (like web servers) that just need to bind on a port below 1024 do not have to run as root: they can just be granted the net_bind_service capability instead. And there are many other capabilities, for almost all the specific areas where root privileges are usually needed.
>
> This means a lot for container security. Let's see why!
>
> Your average server (bare metal or virtual machine) needs to run a bunch of processes as root. Those typically include SSH, Cron, syslogd; hardware management tools (e.g., load modules), network configuration tools (e.g., to handle DHCP, WPA, or VPNs), and much more. A container is very different because almost all of those tasks are handled by the infrastructure around the container:
>
> SSH access will typically be managed by a single server running on the Docker host;
>
> Cron, when necessary, should run as a user process, dedicated and tailored for the app that needs its scheduling service, rather than as a platform-wide facility;
>
> log management will also typically be handed to Docker, or by third-party services like Loggly or Splunk;

hardware management is irrelevant, meaning that you never need to run udevd or equivalent daemons within containers;

network management happens outside of the containers, enforcing separation of concerns as much as possible, meaning that a container should never need to perform ifconfig, route, or ip commands (except when a container is specifically engineered to behave like a router or firewall, of course).

This means that in most cases, containers will not need "real" root privileges at all. And therefore, containers can run with a reduced capability set; meaning that "root" within a container has much fewer privileges than the real "root". For instance, it is possible to:

deny all "mount" operations;

deny access to raw sockets (to prevent packet spoofing);

deny access to some filesystem operations, like creating new device nodes, changing the owner of files, or altering attributes (including the immutable flag);

deny module loading;

and many others.

This means that even if an intruder manages to escalate to root within a container, it will be much harder to do serious damage or to escalate to the host.

This won't affect regular web apps, but malicious users will find that the arsenal at their disposal has shrunk considerably! By default, Docker drops all capabilities except those needed, a whitelist instead of a blacklist approach. You can see a full list of available capabilities in Linux manpages.

One primary risk with running Docker containers is that the default set of capabilities and mounts given to a container may provide incomplete isolation, either independently, or when used in combination with kernel vulnerabilities.

Docker supports the addition and removal of capabilities, allowing use of a non-default profile. This may make Docker more secure through capability removal, or less secure through the addition of capabilities. **The best practice for users would be to remove all capabilities except those explicitly required for their processes.**

Docker Security – cgroups

Prevent DOS attacks/conditions by limiting resource usage in a container

-c or –-cpu-shares	CPU share
--cpu-period	CPU CFS period
--cpuset	limit which CPUs/cores
--cpu-quota	limit CPU usage
-m or --memory	memory limit
--memory-swap	swap space limit
--kernel-memory	kernel memory limit
--blkio-weight	limits block IO bandwidth

You can limit CPU and RAM use for a container to prevent runaway processes from DOSing the host, using Control groups (cgroups: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt).

From the Docker site (https://docs.docker.com/engine/articles/security/):

> Control Groups are another key component of Linux Containers. They implement resource accounting and limiting. They provide many useful metrics, but they also help ensure that each container gets its fair share of memory, CPU, disk I/O; and, more importantly, that a single container cannot bring the system down by exhausting one of those resources.
>
> So, while they do not play a role in preventing one container from accessing or affecting the data and processes of another container, they are essential to fend off some denial-of-service attacks. They are particularly important on multi-tenant platforms, like public and private PaaS, to guarantee a consistent uptime (and performance) even when some applications start to misbehave."
>
> Some Control Groups resource limits are specified through runtime switches when running Docker:

-c or –-cpu-shares	CPU share
--cpu-period	CPU CFS period
--cpuset	limit which CPUs/cores
--cpu-quota	limit CPU usage
-m or --memory	memory limit
--memory-swap	swap space limit
--kernel-memory	kernel memory limit
--blkio-weight	limits block IO bandwidth

A good blog post on resource management in Docker (from 2014):
https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/

Docker – Seccomp

- Secure Computing Mode (Seccomp) provides a syscall firewall between user-level processes and the Linux kernel
- Support and a default profile is implemented as of Docker Engine 1.10
- Whitelist or blacklist filters define what system calls and arguments are allowed
- Default filter blocks over 50 calls when a container runs as non-privileged user

Seccomp (Secure Computing Mode) allows a process to specify filters which allow, trace or block system calls (and parameters) before the kernel. It acts like a syscall firewall, to limit/filter what is allowed. This is based on the same approach as the Google Chrome sandbox and provides similar protection.

Seccomp filters were added to Docker Engine 1.10, using a profile which defines syscalls and filters for these calls. You can define whitelists (by denying all calls by default and then listing only those calls that you want to allow), or a blacklist of denied calls.

A default filter is used if the container is not run as privileged (https://github.com/docker/docker/pull/18780). It blocks over 50 calls by default, using a blacklist. You can run containers with --security-opt seccomp:unconfined if you need to run without any seccomp profile

More information on seccomp profiles and which calls are filtered by the default profile can be found here: https://github.com/docker/docker/blob/master/docs/security/seccomp.md

Here is a demo showing seccomp in action: https://asciinema.org/a/2rmol1dao8qf7tab5gcudq9ck

Security Profiles for Docker

High-level interface to describe security requirements for different containers

- Seccomp filters (whitelist and blacklist)
- Linux Security Model (LSM) definitions (AppArmor, SELinux, etc.)
- Limit capabilities
- Limit resource use (ulimits, cgroups)

Security Profiles are still under development:

- Seccomp profiles implemented in Docker 1.10
- Project Bane: AppArmor profiles

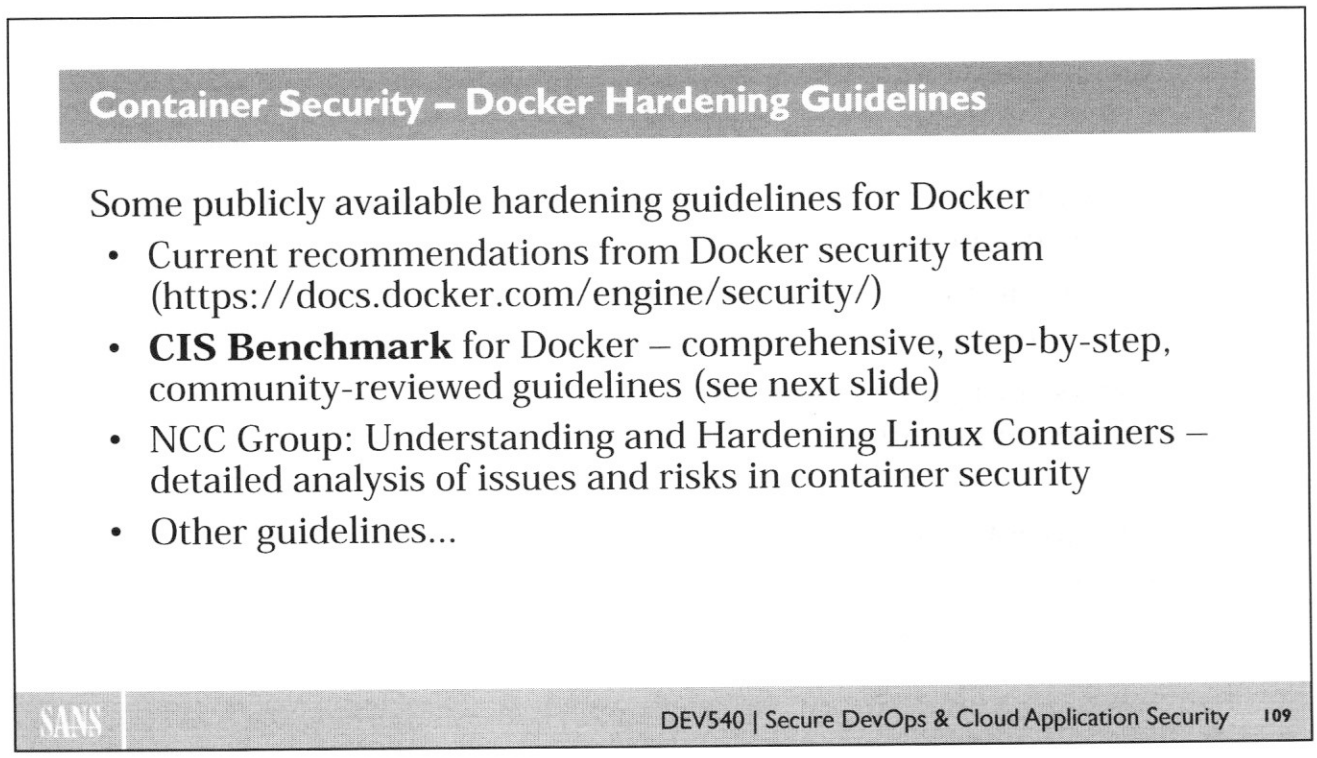
Security Profiles for Docker

Security Profiles are a way to describe the security requirements and ship it with the container – you can create tailored profiles for different containers, with fine-grained definitions. The goal is to provide a high-level, abstracted configuration interface for security containment controls, including seccomp filters (whitelisting and blacklisting) LSM definitions (Linux Security Modules – AppArmor, SELinux, etc.), ulimits, and capabilities.

As of Sept 2015, support is available in runC.

The current description of this feature (Feb 2016) can be read here:
https://github.com/docker/docker/issues/17142#issuecomment-148974642

The first steps in implementing security profiles for Docker are the seccomp profiles implemented in Docker 1.10, and Project Bane, which implements AppArmor profiles for Docker: https://github.com/jfrazelle/bane



Docker Security Guidelines

Docker's security team provides a set of basic steps and recommendations to take for securing your Docker installation: https://docs.docker.com/engine/security/

NCC Group: Understanding and Hardening Linux Containers

A comprehensive analysis by Aaron Grattafiori, NCC Group (April 2016) of Linux container technology including LXC, rkt and Docker, that looks at how containers work, general security issues with containers, and general and specific risks and hardening guidelines for containers.

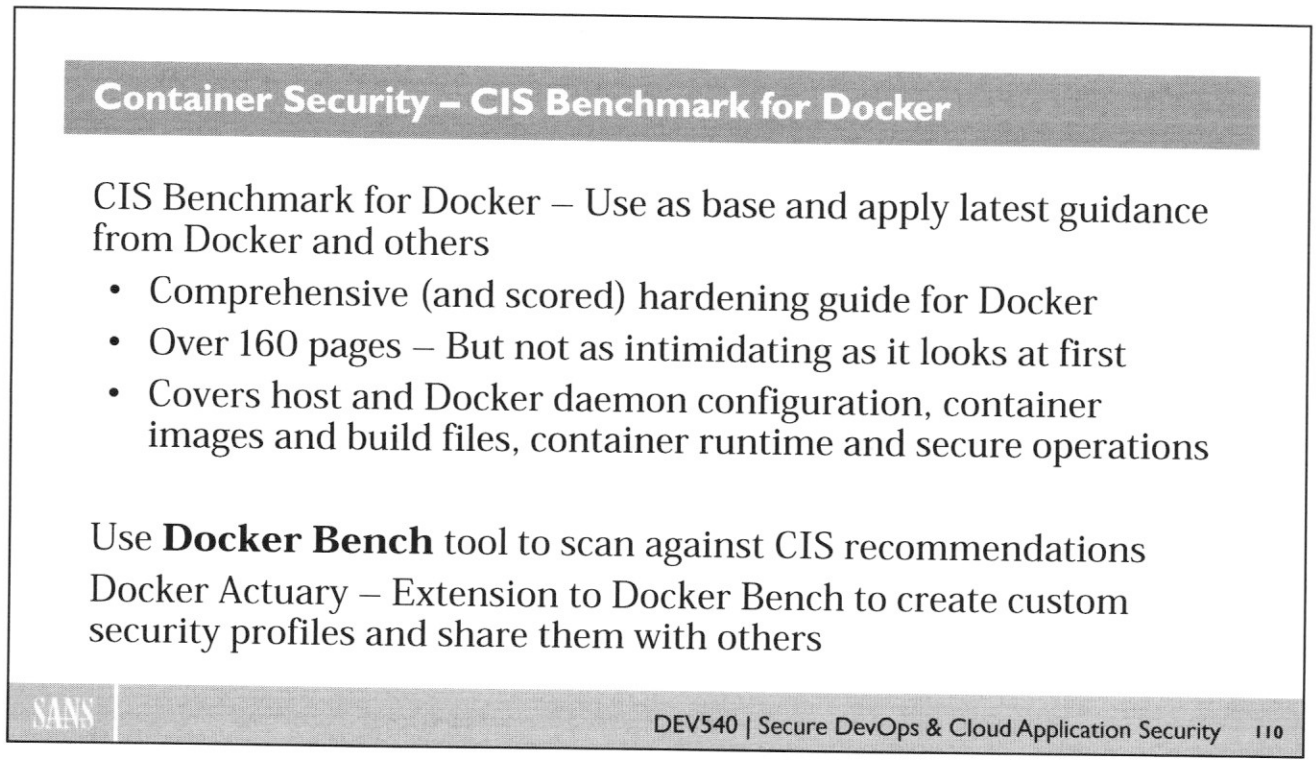
https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-10pdf/

Other Docker Hardening Guidelines

"Security Best Practices for Building Docker Images" by Michael Boelen: http://linux-audit.com/security-best-practices-for-building-docker-images/

"Gotham Digital Security Docker Secure Deployment Guidelines" (for Docker 1.4): https://github.com/GDSSecurity/Docker-Secure-Deployment-Guidelines

"Field Guide to Docker Security": https://zwischenzugs.wordpress.com/2015/05/21/a-field-guide-to-docker-security-measures/



CIS Benchmark Guide for Docker (1.13)
https://benchmarks.cisecurity.org/tools2/docker/CIS_Docker_1.13.0_Benchmark_v1.0.0.pdf
The Center for Internet Security has published a benchmark for hardening Docker, which is regularly updated. This is the most complete and comprehensive guide for securing Docker. Use it as a base and apply the latest guidance from Docker and the community until an updated benchmark is released.

The CIS benchmark covers:

- Basic steps for hardening the host configuration
- Ensuring that the kernel and Docker are up to date
- Auditing the Docker daemon and docker configuration files
- Configuring limits for the Docker daemon, restricting network activity between containers
- Permissions on Docker daemon configuration files
- Security issues around container images
- Container runtime security
- Secure Docker operations guidelines

Docker Bench for Security (https://github.com/docker/docker-bench-security)
Docker Bench (https://dockerbench.com/) is a bash script from Docker that checks for common best practices in Docker configuration, based on the CIS Benchmark. It is packaged as a pre-built Docker container that introspects other container and warns you about vulnerabilities.

Docker Actuary (https://github.com/diogomonica/actuary)
Docker Actuary is an extension to Docker Bench, developed in Golang, which can be extended use different security profiles (so that you can define your own guidelines, or create profiles that can be shared with others).

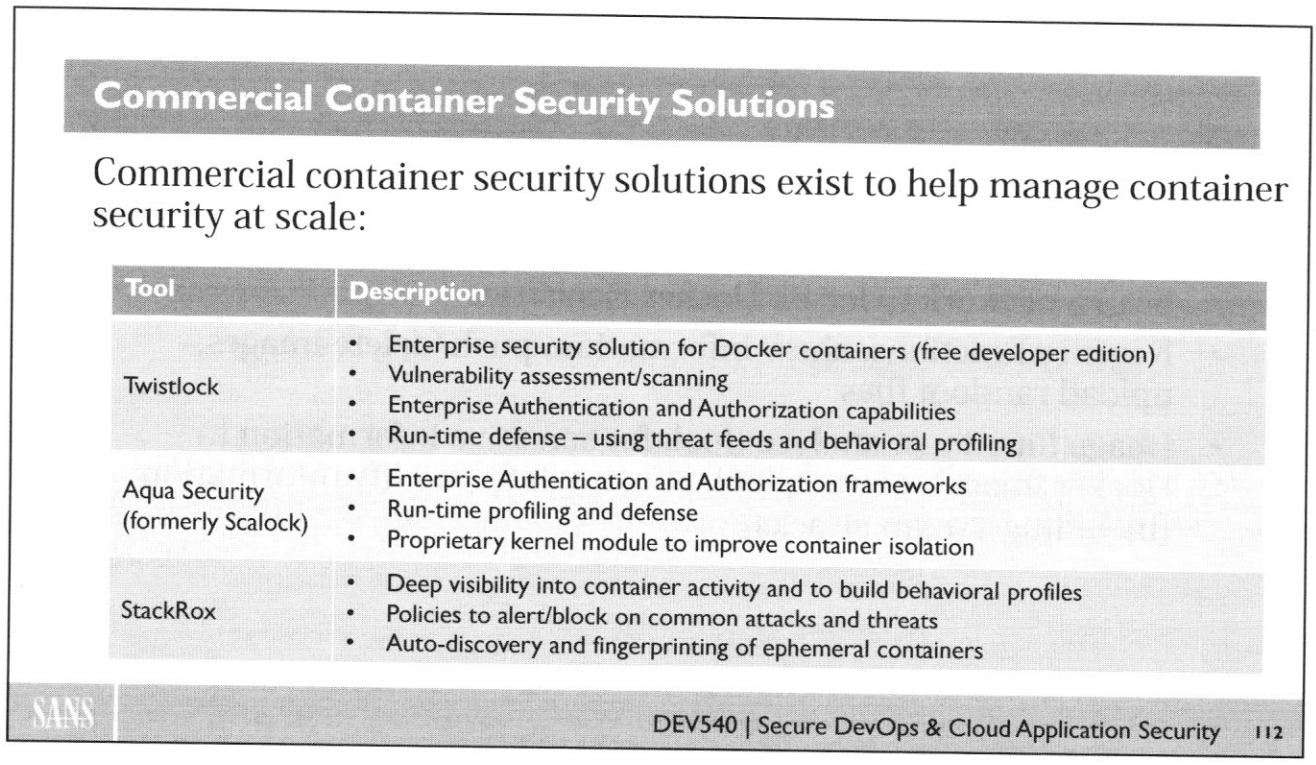
Container Security – Attacking Docker

DockerScan project – open source analysis and pen testing attack toolset for Docker

- Scan a network to locate Docker Registries
- Registry functions: show information, push/delete images, upload random files
- Image functions: analyze (look for sensitive information in Docker image), extract, look at meta data, modify information (including Trojan attacks)

DockerScan (https://github.com/cr0hn/dockerscan) is an early stage open source pen testing toolset for dissecting and attacking Docker implementations (Docker registries and images. It includes commands to scan for Docker registries and play with registry content, and to scan Docker images for sensitive information (passwords, user and IP information), extract images and conduct automated attacks.

See: https://www.slideshare.net/cr0hn/rootedcon-2017-docker-might-not-be-your-friend-trojanizing-docker-images/



Defending containers can be difficult, especially with the container sprawl problem mentioned previously. There is no easy way to know how many out-of-date or misconfigured containers you have running at any time, particularly because containers are ephemeral. There are a handful of commercial solutions to help deal with these issues, all providing similar capabilities:

Twistlock (https://www.twistlock.com/)
Twistlock provides run-time monitoring and defense for Docker containers. You install a Twistlock Container Defender on every Docker Host. The Container Defender looks inside all of the other containers on the system and compares the runtime configuration against policy templates (based on best practices like the CIS benchmark), vulnerability feeds and automatic profiling policies to detect or block problems.

Twistlock has a few different parts:

1. Open source framework for developers to enforce quality gates on containers before they are pushed out, including scanning for vulnerabilities in packages and in configuration.
2. Enterprise management framework for central monitoring of containers and enforcing of policies – deep inspection, detect and prevent vulnerabilities and threats.
3. Fine-grained access control – teams can only manage their apps (test or production), using access control restrictions defined in Kerberos and LDAP. Twistlock engineers helped develop the authorization plug-in framework released by Docker.
4. Enterprise authentication for access to Docker containers.

Twistlock scans images for known vulnerabilities using a live threat intelligence stream, as well as checking against configuration best practices (including CIS guidelines). During the scan, it builds up information on the composition and intent of each container – to define its expected behavior, what actions should be expected in each container, and what should not be allowed.

At runtime, Twistlock provides a range of defensive capabilities:

1. Enforcing configuration policies
2. Monitoring container memory and storage for anomalies
3. Whitelisting and blacklisting processes
4. Blocking malicious networks using the live threat intelligence stream
5. Identifying – and blocking – attempts to change container configuration at runtime
6. Alerting or blocking on other malicious/unexpected actions, based on the profile of the container

Aqua Security fka Scalock (https://www.aquasec.com/)

An enterprise security solution for Docker, CoreOS rkt, VMware, and Microsoft Windows. The Aqua Container Security Platform runs in a master/agent model: agent containers run in privileged mode on all managed Docker hosts. Agents check server/container configuration using Docker Bench and scan containers and images for vulnerabilities and out-of-date packages and libraries. Like Twistlock, Aqua Security provides runtime defense to protect containers from other containers or attackers:

- Fine-grained, role-based user access control (can set up restrictions based on containers, hosts, and applications)
- Audit trail of container activity, user access, and host configuration changes
- Automatically enforces security policies at runtime

StackRox (https://www.stackrox.com/)

StackRox is another runtime container defense solution, which provides deep visibility into what is happening in every container. It uses auto-discovery and fingerprinting techniques to identify and profile containers as they start up, and tools to visualize container activity. StackRox provides a library of attack profiles and data filters, and uses machine learning to identify problems and automatically block attacks. StackRox also provides deep vulnerability scanning for images.

Container Security – CoreOS rkt

CoreOS offers an alternative container: rkt aka Rocket

- CoreOS (an early supporter of Docker) launched a competing, lightweight container called rkt in December 2014
- rkt is designed to work with existing tools and platforms and is simpler than Docker
- rkt can run Docker and Appc images
- Designed with security/production in mind

rkt (pronounced "rock-it") from CoreOS is a leading alternative container to Docker.

CoreOS provides a minimal Linux operating system with built-in clustering support, minimal binaries, and no packaging system. It is dependent on containers for managing software and apps.

CoreOS was one of the early supporters of Docker, but launched rkt in December 2014 because they felt that Docker had moved from providing a simple, reusable standard container to "building tools for launching cloud servers, systems for clustering, and a wide range of functions: building images, running images, uploading, downloading, and eventually even overlay networking, all compiled into one monolithic binary running primarily as root on your server… It is not becoming the simple composable building block we had envisioned. " https://coreos.com/blog/rocket/

rkt (https://github.com/coreos/rkt) was designed on a new Application Container (Appc) specification that tries to improve on the original Docker V1 specification.

rkt is "designed to be composable, secure, and fast". Some of rkt's key features and goals include:

- First-class integration with init systems (systemd, upstart) and cluster orchestration tools (fleet, Kubernetes, Nomad)
- Compatibility with other container software (e.g. rkt can run Docker images)
- Modular and extensible architecture (network configuration plugins, swappable execution engines based on systemd or KVM)

rkt 1.0 was released in February 2016. It offers a range of security features including VM-based container isolation, SELinux support, TPM integration, image signature validation, and privilege separation. A series of incremental releases since then introduce seccomp filtering (1.12), improved isolators (1.6, 1.7), etc.

In June 2015, CoreOS and Docker (together with Google, Red Hat and other enterprise software organizations) announced that they were working together to define a standard container format under the Open Container Initiative: (https://www.opencontainers.org/). The new standard combines Docker's LibContainer and CoreOS' App Container (Appc) to create a unified container image format. Both Docker and rkt support the new format. https://coreos.com/blog/app-container-and-the-open-container-project/

Comparing rkt to Docker and other containers:

https://labs.ctl.io/interviews/what-is-rocket-and-how-its-different-than-docker/

https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html

The Docker Engine is an application container runtime implemented as a central monolithic API daemon. Docker can resolve a "Docker Image" name, such as quay.io/coreos/etcd, and download, execute, and monitor the application container. Functionally, this is all similar to rkt; however, along with "Docker Images", rkt can also download and run "App Container Images" (ACIs) specified by the App Container Specification (Appc).

Besides also supporting ACIs, rkt has a substantially different architecture that is designed with composability and security in mind.

Process Model

At the time of writing, the Docker Engine daemon downloads container images, launches container processes, exposes a remote API, and acts as a log collection daemon, all in a centralized process running as root. While such a centralized architecture is convenient for deployment, it does not follow best practices for Unix process and privilege separation; further, it makes Docker difficult to properly integrate with Linux init systems such as upstart and systemd. Since running a Docker container from the command line (i.e. using docker run) just talks to the Docker daemon API, which is in turn responsible for creating the container, init systems are unable to directly track the life of the actual container process.

rkt has no centralized "init" daemon; instead, launching containers directly from client commands, making it compatible with init systems such as systemd, upstart, and others.

Privilege Separation

rkt uses standard Unix group permissions to allow privilege separation between different operations. Once the rkt data directory is correctly set up (for example, by rkt install), the container image downloads and signature verification can run as a non-privileged user.

Lab #2.2 – Audit Docker's Security

Estimated Time: 20 Minutes

Use the Docker Security Benchmark to assess and improve the state of our lab VM.

OBJECTIVES

Audit the security of the lab VM using the CIS Docker Benchmark

Enable Docker's authorization functionality

PREPARATION

Check out the Docker security benchmark project from GitLab

This page intentionally left blank.

Lab #2.2 Summary – Audit Docker's Security

In this lab, you ...

- Assessed the security of a Docker environment using the Docker Security Benchmark
- Used puppet to reconfigure Docker
- Remediated one of the weaknesses found
- Verified the functionality of Docker's authorization controls

After reviewing the results of the Docker Security Benchmark, we made a significant change to our Docker configuration: we enabled authorization. In doing so, we were able to see how specific actions could be allowed or denied individually.

We also observed that the version of Docker in use for the lab VM does not capture user identity from the certificate used to authenticate to the Docker service. This has been fixed in a newer release. As mentioned previously, major investments are being made in Docker's security.

Docker Security Summary

- ✓ Don't depend on Docker defaults – Docker provides some built-in architectural sandboxing and other runtime protection, but is NOT secure by default
- ✓ Make appropriate trade-offs between developer flexibility and security
- ✓ Major investments are being made in security today – expect improvements from Docker, the community, and vendors

Basics of security for Docker:

1. Only use container images that you trust
2. Regularly scan for vulnerabilities in containers and patch them
3. Do not run as root – or at least enable user namespacing (as of 1.10)
4. Drop capabilities wherever possible and take advantage of seccomp to limit syscalls
5. Keep the Docker Engine up-to-date with the latest version to take advantage of improving security capabilities
6. Take advantage of host O/S security features and make sure to harden the O/S environment
7. Follow the CIS hardening guidelines to the extent possible – use Docker Bench to audit
8. Do not install Docker in production using default configs, and monitor production use carefully. Consider the use of enterprise defense technologies such as Twistlock.
9. Recognize that the platform and tooling are still immature and rapidly changing. Expect that there will be gaps and bugs for some time
10. Follow #dockersecurity on Twitter

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- *Monitoring and Metrics*
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.

Monitoring and Metrics in DevOps

DevOps feedback loops encourage extensive monitoring of production (and build) environments to help teams make data-driven decisions

- Measure (and Graph) Everything – Etsy
- DevOps monitoring technology stack:
  Statsd, logster, **graphite**, **grafana,** seyren
- Monitoring Sucks and Monitorama are community initiatives to improve monitoring practices and tools
- Monitoring as a Service: New Relic, Datadog, Stackify, ...

Monitoring is an important part of DevOps, by building feedback loops from production back into engineering. This includes:

- Measuring performance and latency at different layers and granularity, from Real User Monitoring using synthetic transactions (https://en.wikipedia.org/wiki/Real_user_monitoring) to resource usage and latency timing
- Reporting and tracking runtime errors and exceptions to detect – and understand – operational problems
- Collecting usage statistics and conversion statistics in online experiments to evaluate the success of new features, workflows, and user experience options (A/B testing)
- Tracking changes, and the impact of changes, by correlating runtime statistics and exceptions with the timing of changes – see how this is done at Etsy (https://codeascraft.com/2010/12/08/track-every-release/)
- Identifying and tracking other metrics that are important to the business, operations, InfoSec and developers

Etsy has taken the idea of monitoring and measuring to the extreme, in what they call "Measure Anything, Measure Everything" (https://codeascraft.com/2011/02/15/measure-anything-measure-everything/). They capture and measure thousands of different application and system and operational metrics, which are recorded in a time-series roll-up database, visualized on dashboards, compared against trends, and correlated with other events.

Monitoring Technology Stack

Operations data written to log files can be captured and analyzed using tools like logstash or Splunk,

Etsy and others have collaborated on an open source monitoring stack which is commonly used in DevOps environments:

- Statsd (https://github.com/etsy/statsd) – Libraries and a collector for publishing and collecting runtime metrics in a lightweight manner
- Graphite (http://graphite.readthedocs.org/en/latest/) – A data management platform for storing and managing time series metrics and rendering these metrics in graphs
- Logster (https://github.com/etsy/logster) – Tails log files, parses and grabs metrics data, publishes metrics to statsd – can be used to collect metrics from OS, network and outside apps. Note that other open source log management tools like logstash and nxlog also have output plugins for statsd.
- Grafana (https://github.com/grafana/grafana) – A tool for building online metrics dashboards on top of graphite
- Seyren (https://github.com/scobal/seyren) – Alerting tool on top of graphite – set warn/error thresholds for metrics

There is an extensive ecosystem of tools that work with/on graphite metrics: http://graphite.readthedocs.org/en/latest/tools.html

Docker image for easily setting up graphite and statsd: https://github.com/hopsoft/docker-graphite-statsd

Monitoring Sucks... and Monitorama

"Monitoring Sucks" (https://github.com/monitoringsucks) is a movement to critically review and improve the state of operations monitoring. You can learn more about this at http://lusislog.blogspot.ca/2011/06/why-monitoring-sucks.html

And by following the Twitter tag: #monitoringsucks

There is also a conference dedicated to the problems of monitoring and open source monitoring solutions: http://monitorama.com/

Monitoring as a Service

A number of companies provide cloud-based "Monitoring as a Service":

- New Relic http://newrelic.com/solutions/production-monitoring
- Ruxit https://ruxit.com/
- Stackify http://stackify.com/
- DataDog https://www.datadoghq.com/
- Amazon CloudWatch for AWS users https://aws.amazon.com/cloudwatch/

Metrics – Data Formats

Two popular formats for sending metrics data: statsd, graphite. Use case drives selection of appropriate tool

Format	statsd	graphite
Protocol	UDP	TCP
Time	All measurements happen "now"	Measurement includes timestamp
Data Types	Gauge, Counter, Timer, Histogram, Meter	Value (floating point number)
Format	<metric.path>:<value>|<type>	<metric.path> <value> <timestamp>
Aggregate	At server, based on type, configuration	At source. Same path + timestamp is overwritten at server

Two popular formats for delivering metrics to a collection system are statsd and graphite. Both use a plain text string to carry the data, making it trivial to generate metrics, even from command line applications. Also, both use a '.' separated metric path to help organize the data. Key differences in the two formats are summarized in the table above.

Here are examples of how an application might format a metric used to count registrations for this course

Statsd: sans.registration.class.dev.534:1|c

Graphite: sans.registration.class.dev.534 1 1486929875

For web applications, it is common to use the statsd format to export metrics, since the UDP connection is low overhead and non-blocking. Because UDP is unreliable, designing the metrics collection infrastructure to place collectors/aggregators for statsd data close to the sources is also a common pattern. Additionally, using an aggregator for the statsd metrics allows for computation of statistical data about the metrics on-the-fly, before persisting the data – e.g. max, min, mean, Nth percentile, rate, etc.

Note: for both graphite and statsd, the name can refer to either the data format or the service which receives the metric data.

https://github.com/b/statsd_spec

http://graphite.readthedocs.io/en/latest/feeding-carbon.html#the-plaintext-protocol

Metrics – Data Processing

Graphite (the service) has a rich set of functions to manipulate data

- Time Shifting (compare to last hour, 24 hours ago, last week)
- Grouping / summarization / aggregation
- Draw as infinite (to create vertical lines)
- Holt-Winters forecasting
- Map / Reduce / Integral / Derivative / Ratio

Graphite includes a wide range of functions which can be used to analyze the metrics data stored in the system.

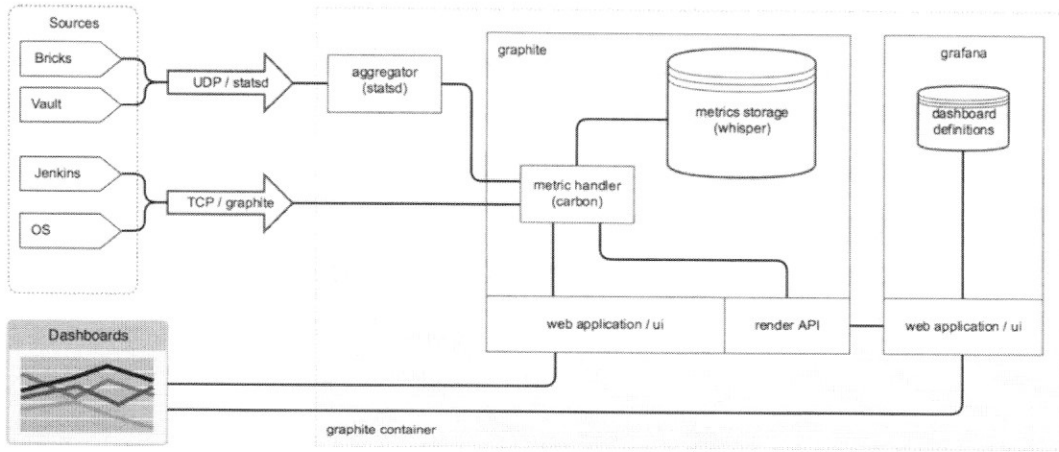
Grafana's metric editor includes quick access to each function and its documentation.

Full documentation of the functions is available online -https://graphite.readthedocs.io/en/latest/functions.html

Must watch video: Christopher Rimondi "Using DevOps Monitoring Tools to Increase Security Visibility" https://www.youtube.com/watch?v=TNCVv9itQf4

Covers some excellent techniques for applying graphite functions to data, to get actionable results

Metrics – Data Flow



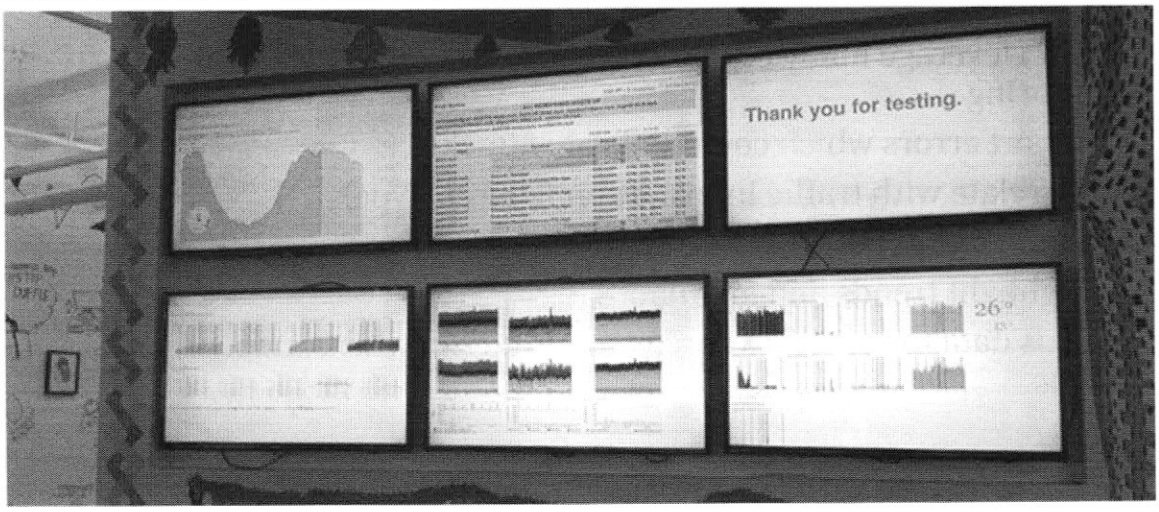
In our lab environment, we have many of the applications sending metrics to graphite for collection and storage. We also use Grafana to design and display dashboards looking at the data stored in graphite.

The class VM uses collectd to monitor disk, CPU, memory, and network activity and sends the data directly to graphite. Jenkins is also configured to send data directly to graphite. Both Jenkins and collectd utilize internal subsystems to aggregate activity counts before sending the data to graphite.

On the other hand, both Vault and the Bricks containers are configured to send metrics using the statsd format, which are then aggregated and sent to graphite for storage.

This general data flow is a common pattern, although the implementation in the lab VM does not address needs like scalability and redundancy.

Production Monitoring at Etsy



Sample of a runtime monitoring wall at Etsy (courtesy of Zane Lackey). Each team (and each developer) can set up their own monitoring displays, to watch for what they consider important at the time.

Features of note:

- Vertical lines on bottom left and bottom right monitors: likely code releases or similar event
- Top left monitor: time shift overlay

Monitoring and Metrics – Security

You can leverage many of the same tools and approaches for security monitoring

- Report errors which could indicate attacks
- Correlate with traffic information (source, type)
- Graph and report this information
- Highlight trends and anomalies

Use this data to identify where you are at risk, where to focus your testing

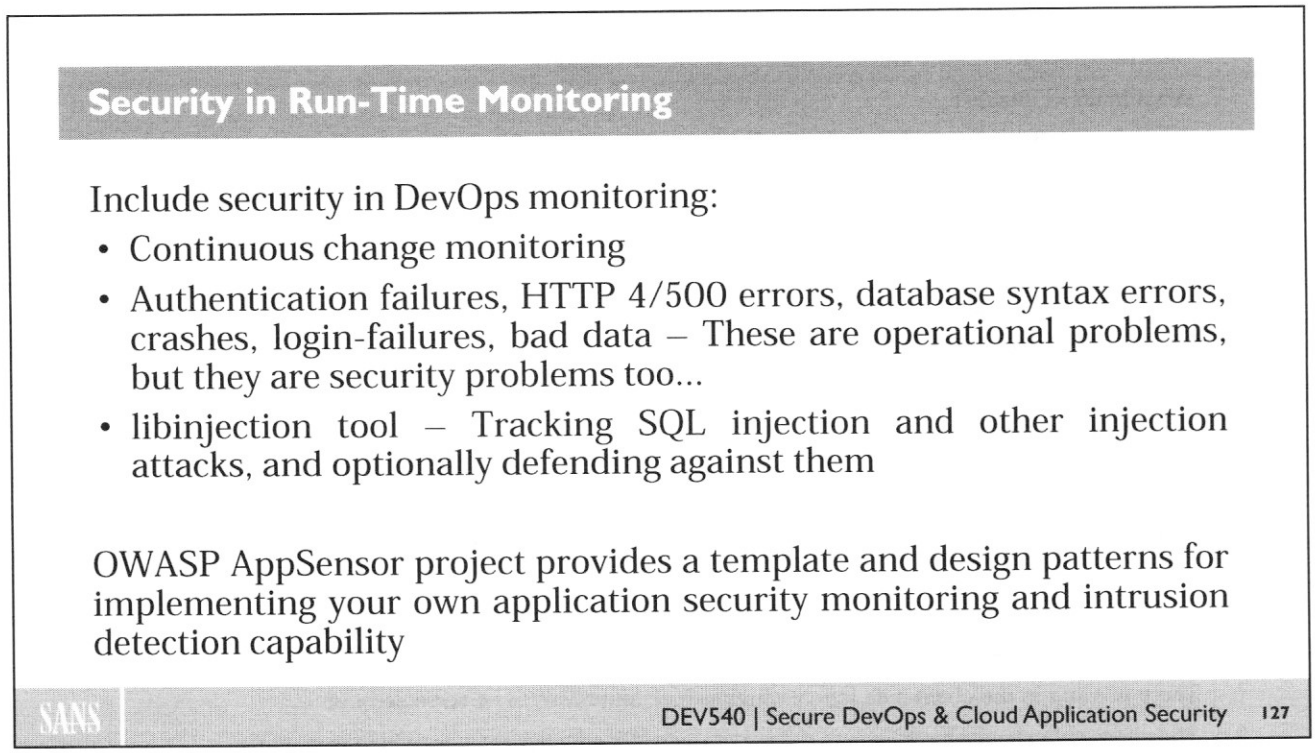
Run-time security monitoring can be integrated into operations monitoring, taking advantage of many of the same tools (statsd, graphite, grafana, logster…) and the same approaches:

- Collecting detailed runtime metrics and statistics to increase visibility into security risks and attacks
- Correlate with traffic information (source, type, rate of traffic)
- Graphing data to identify trends and highlight anomalies, and making this data available to anyone on the team
- Using this data to determine where you are most at risk, where to focus your testing

Must watch: Christopher Rimondi "Using DevOps Monitoring Tools to Increase Security Visibility" https://www.youtube.com/watch?v=TNCVv9itQf4

Read these posts by Rimondi on Anomaly Detection using Graphite:

- http://www.securitygrit.com/2015/07/anomaly-detection-with-holt-winters-in.html
- http://www.securitygrit.com/2015/07/anomaly-detection-with-timeshift-in.html
- http://www.securitygrit.com/2015/07/anomaly-detection-with-coefficient-of.html



Don't leave responsibility for security monitoring to the SOC. Security should be part of the day-to-day operational monitoring of the application.

It's easier to identify anomalies in the context of operations. Operations is watching for anomalies already. Security attacks are anomalies. Therefore…

As part of the principle of "You Built it, You Run it", developers should be provided read access to application logs and dashboards in production. The people who wrote, and help support, the application, should have a better idea of what normal behavior in the system should look like, and spot exceptions and anomalies—including security attacks.

Security-related events and metrics should be available to everyone on the team.

Some security-related events and exceptions include:

- Changes to code and system configuration
- Changes to sensitive files, and changes to permissions on files/directories
- Changes to credentials and secrets
- Access to sensitive logs
- Authentication failures—particularly, spikes in rate of change (a handful of occasional authentication failures are not interesting, but spikes or clusters are)
- HTTP 4xx/5xx errors
- Bad or unusual SQL statements
- Run-time crashes
- Bad data caught by server-side validation checks

By tracking security-related events and exceptions you can look for exceptions and trends:

- How many security attacks are you seeing? Is this increasing?
- What kinds of attacks?
- What part of the system is being attacked? What URLs, APIs, files, …
- Where are the bad guys attacking the system from? (tie into network security monitoring)
- Are the attacks succeeding?

Look for trends and anomalies in this data. As with any operations monitoring, be prepared to iterate and continuously review to filter out false positives.

OWASP AppSensor project

The OWASP AppSensor Project (https://www.owasp.org/index.php/OWASP_AppSensor_Project) explains how to embed intrusion detection/control into the application at different points in the application architecture to detect – and potentially block – attacks in progress. Where and how to add security checks in an application.

For example, an application enforces data validation checks at the client-side for immediate feedback, and at the server-side as a good practice defense-in-depth control. If the server-side validators fire, this means that

1. There is a bug in the client-side logic (which is bad, and needs to be corrected); or
2. The client-side checks have been bypassed or overridden by an attacker (which is worse).

It is important to decide what action to take: alert operations, or, if the risk is high enough, it may be necessary to block subsequent requests to the server until someone investigates and resolves the situation.

Libinjection

Libinjection (https://github.com/client9/libinjection/) is a tool to track SQL injection attempts at runtime – and potentially block them, from Nick Galbreath http://www.slideshare.net/codeblue_jp/nick-galbreath-enpub

Attack-Driven Defense

Use production attack data and exceptions to create feedback loops from production back to engineering

- Recognize that your system is – or will be – under attack and take advantage of the information that this will provide you
- Attack data from production tells you where you need to have strong defenses
- Where do you need to prioritize your reviews, testing?
- Are you improving the defensibility of the system?
- These attacks aren't theoretical – they are in progress!

Move to the left of the Cyber Kill Chain and catch attackers early

Security data from production – information on potential attacks and exceptions – can be used for what Zane Lackey (formerly Etsy, now Signal Sciences) calls "Attack-Driven Defense" (http://www.slideshare.net/zanelackey/attackdriven-defense).

Recognize that your system is – or will be – under continuous attack. Prepare for this – and take advantage of the information that this will provide you. Like data-driven A/B testing, information on real or potential attacks can serve as an important feedback loop to the security team and to the engineering teams. This is concrete data on real threats – not hypotheticals.

Surface this information back to engineering so that they understand it. The more that you measure, and the more transparent you are with this information, the more data that people will have to recognize, understand and solve problems.

Use real metrics on attack data in production to drive your security program priorities, to help you – and the engineering teams - decide where you need to focus your testing and reviews and remediation.

Use the feedback loops to measure which changes are improving the defensibility of the system. Watch for changes (positive and negative) after deployments.

Continuously monitor status in order to move closer to the left (the start) of the Cyber Kill Chain. This is a model taken by Lockheed Martin from military tactics, to describe the steps in a security intrusion and how to defend against an attacker. By moving left, closer to the start of the Cyber Kill Chain, you try to catch an attacker before they have gained a foothold on your system.

The Cyber Kill Chain is outlined here: https://cyber.leidos.com/solutions/cyber-kill-chain

The original Lockheed-Martin paper can be found at
http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf

1. Reconnaissance
2. Weaponization
3. Delivery
4. Exploitation
5. Installation
6. Command and Control
7. Actions on Objectives

Signal Sciences – SaaS platform for Web Security Visibility

Run-time security monitoring/firewall service for DevOps/WebOps environments

- SaaS backend
- Built up from ideas developed at Etsy
- Provides runtime visibility of security anomalies and ability to block attacks based on threat priority
- Automated feedback loops back into development and operations

Signal Sciences (https://www.signalsciences.com/product/) is a startup which offers a next-generation SaaS-based application firewall for online systems. It sets out to "Make security visible" by providing increased transparency into attacks, and the ability to identify anomalies and block attacks.

Signal Sciences was started by a group of DevOps Security leaders, including the former heads of security and of operations engineering at Etsy (Zane Lackey and Nick Galbreath). The firewall takes advantage of ideas and techniques that they developed for building effective feedback loops from production back to operations and development. Instead of relying on signatures to detect attacks, it inspects and tokenizes traffic and correlates it with other signals such as runtime errors to prioritize threats and determine when an attack should be blocked.

Attack information is displayed on dashboards for trending and correlation, and people can be notified through email or PagerDuty, or through collaboration services such as Slack or HipChat. APIs are available to integrate alerts into tools like Splunk, Jira, and ThreadFix.

Cloud Security Monitoring and Run-time Defense

Tools/services to monitor/enforce security of cloud applications:

- Alert Logic
- Amazon AWS Inspector, CloudTrail, Trusted Advisor
- CloudPassage Halo
- Dome9 SecOps
- Evident.io
- Illumio
- Netflix Security Monkey, Conformity Monkey
- Palerra LORIC
- Threat Stack

There are a number of tools and services available to monitor the security of cloud environments and applications running in the cloud, as well as providing runtime protection including micro-segmentation and host firewall management, centralized configuration management and policy enforcement, and file integrity checking. Some of these solutions work across multiple cloud platforms (public, private and hybrid).

Alert Logic (https://www.alertlogic.com/) SaaS solution providing managed IDS, log management/SIEM, WAF, data analytics and monitoring of AWS environments.

Amazon AWS Inspector (https://aws.amazon.com/inspector/) Automated security assessment of applications deployed on AWS, scans for vulnerabilities or deviations from best practices. Includes rules for PCI DSS and other compliance standards. It provides a prioritized list of security issues and recommendations on how to fix them. As of Feb 2016, in limited Preview release only.

Amazon AWS CloudTrail (https://aws.amazon.com/cloudtrail/) Amazon AWS service that records and logs AWS API calls.

Amazon AWS Trusted Advisor (https://aws.amazon.com/premiumsupport/trustedadvisor/) Checks configuration best practices (cost, security, fault tolerance, and performance) for the deployment of Amazon EC2, Elastic Load Balancing, Amazon EBS, Amazon S3, Auto Scaling, AWS Identity and Access Management, Amazon RDS, Amazon Route 53, and other services.

CloudPassage Halo (https://www.cloudpassage.com/products/) Run-time security and compliance monitoring and runtime protection for Microsoft Azure, AWS, Rackspace and other cloud services. Includes configuration security monitoring, software vulnerability assessment, file integrity monitoring, intrusion detection, multi-factor authentication, server access management, and dynamic firewall management.

Dome9 SecOps (https://dome9.com/) API-level and agent-based security protection for AWS: firewall management, runtime configuration checking, file integrity monitoring, tamper protection, dynamic access leasing, multi-factor authentication, and compliance auditing.

Evident.io (http://evident.io/) Continuous security monitoring of AWS accounts for known security risks.

illumio (illumio.com) workload security monitoring and policy enforcement for data centers and cloud environments.

Netflix Security Monkey (http://techblog.netflix.com/2014/06/announcing-security-monkey-aws-security.html) Open source, automated runtime checking of security-related AWS components and configuration items, including security groups, S3 bucket policies, and IAM users. Monitors changes to configurations, and audits settings against a set of configuration rules (built-in rules can be extended).

Netflix Conformity Monkey (http://techblog.netflix.com/2013/05/conformity-monkey-keeping-your-cloud.html) Open source, automated checking of AWS configurations against best/safe practices

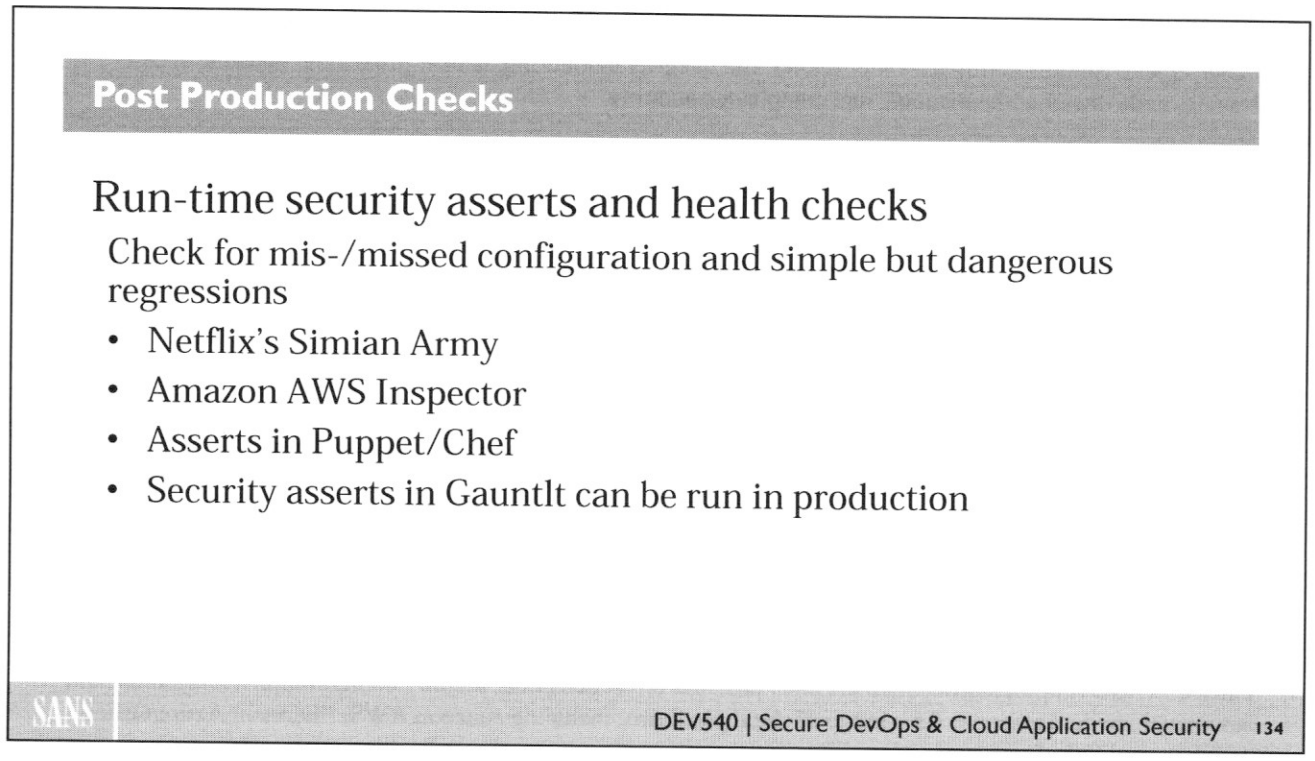
Palerra LORIC (http://palerra.com/platform/) provides threat detection, security configuration management and automated incident response for a range of cloud platforms, including Amazon AWS, Microsoft Azure/Office 365, box, and Salesforce.com.

Threat Stack (https://www.threatstack.com/) Behavioral-based, real-time security monitoring/IDS for AWS instances.

For an overview of the market for cloud security solutions, see

Forrester's report "Market Overview: Cloud Workload Security Management Solutions – Automate or Die", Andras Cser, June 2, 2015

http://blogs.forrester.com/andras_cser/15-06-02-market_overview_cloud_workload_security_management_solutions_automate_or_die



Security testing doesn't stop at deployment. You should also have active post-deployment security tests and checks in production with feedback to operations and developers.

This is especially important in highly dynamic environments where changes are constantly being made by developers because small mistakes could introduce vulnerabilities or operational problems. These asserts build on detective change control and host-based IDS systems like Tripwire, but instead of checking for unauthorized changes, they check to make sure that changes are implemented correctly.

These double-checks can include:

- Checking runtime versions of components and signatures on code
- Making sure that specific ports are open or closed
- Ensuring that logging and auditing functions are working correctly
- Checking permissions on sensitive files and directories
- Checking that authentication services and rules are working correctly
- Ensuring that SSL is configured safely

Create a basic set of checks and smoke tests which ensure that the system is set up correctly and safely – if these tests fail, then the deployment should be rolled back. These checks and asserts can be done using automated tools like Gauntlt.

Netflix's Security Monkey and Conformity Monkey are examples of runtime checkers for Cloud environments.

Security Monkey
http://techblog.netflix.com/2014/06/announcing-security-monkey-aws-security.html
https://github.com/Netflix/security_monkey

Conformity Monkey
http://techblog.netflix.com/2013/05/conformity-monkey-keeping-your-cloud.html
https://github.com/Netflix/SimianArmy/wiki/Conformity-Home

Amazon Inspector is a similar tool for AWS
https://aws.amazon.com/inspector/

Incident Response

- DevOps teams prepare for production problems through GameDay exercises
- Developers and Ops work together to understand and resolve problems
- Blameless postmortems to review and improve
  - Understand what went wrong, when and why
  - Identify and address root causes: "human error" is NOT a root cause
- Security can participate in these exercises and take advantage of the process frameworks and training
  - Helps developers and ops to respond appropriately to breaches
  - Builds positive relationship between security, development, and ops

DevOps teams prepare for failure and are optimized to minimize Mean Time to Detect/Recover from failures. Organizations like Amazon, Etsy regularly conduct Game Days, where they inject failures into production systems under controlled conditions, and monitor and assess the team's capability to respond to the failures. The exercises are carefully planned in advance, and the failure scenarios are tested to understand the potential impact on production systems – and giving the team a chance to review and fix any problems found in the systems or make improvements to diagnostics or monitoring, in advance of the exercise. The point of the exercise is not to find problems in production – it is to evaluate the team's ability to identify and respond to an incident under production conditions.

Read: "Fault Injection in Production: Making the case for resilience testing", John Allspaw
https://queue.acm.org/detail.cfm?id=2353017

Blameless Postmortems – review, learn, and improve

After the exercise is complete, the team holds a postmortem review to go over the incident, determine if there were any surprises, and identify opportunities to improve. The point of this review is to review what happened, what actions people took, and why they responded the way that they did. What information did they have to make their decisions? What was the outcome? How can this be improved?

By sharing information and asking questions, teams can identify the root cause – or root causes – of problems, and work to address them DevOps teams, following the lead of John Allspaw at Etsy, emphasize that "human error" is not a root cause of a problem: a mistake made by a person or by people is a manifestation of a deeper problem that needs to be corrected.

"Beyond Blame: Learning from Failure and Success", Dave Zwieback
http://shop.oreilly.com/product/0636920033981.do

Leveraging Game Days for Security Incident Response

InfoSec can participate in – and learn from – these exercises, and build on the team's capabilities, practices, and tools to improve the organization's ability to respond to security events:

- InfoSec could leverage the team's existing outage incident response team structure, communication processes, escalation models, etc. for responding to breaches and serious security attacks.
- Build on the culture and processes of Blameless Post mortems to help the organization understand and learn from security incidents.
- Build on automation in DevOps to help deal with security incidents. As an example, see Netflix's FIDO tool for responding to malware (https://github.com/Netflix/Fido).
- Understand how to take advantage of the automated CD pipeline artifacts and audit trails for investigation and for remediation.

Morgue – Tool for Analyzing an Incident

Morgue: open source tool built by Etsy to manage postmortem data

- Helps to understand context of an incident and prepare the team's analysis
- Chronology of events
- Attach images: charts, screenshots, ...
- Attach links
- Pull in relevant IRC channels, forum posts with discussions
- Record and link to remediation actions (Jira tickets)

Ensures that postmortem analysis is done consistently between incidents and across teams – helps people learn

Etsy has built and open sourced an extensible platform for managing postmortems, called Morgue (https://github.com/etsy/morgue). Morgue is a simple web app that can pull information from IRC and Jira, and store data (graphs, links) to help understand the context of an incident and to prepare your team's analysis:

- Create the chronology/timeline of events
- Attach images, charts
- Reference external links
- Pull in relevant IRC channels and forum posts
- Record remediation actions (in Etsy's case, through Jira tickets)

Using a tool like Morgue ensures that postmortem analysis follows a consistent structure and playbook, making it easier for other teams to reference and build on, and easier for auditors to review postmortem reports. It also helps new people and other teams learn how to conduct a postmortem.

Watch this presentation from 2013 on how to use Morgue: https://vimeo.com/77206751

Security War Games and Red Team Fire Drills

- Apply the Game Day approach for operations incident management to security
- Rehearse security incident response and learn about security attacks and how to manage them
  - Red Team: attack-driven approach (Mean time to Compromise)
  - Blue Team: defense-driven approach (Mean time to Detect)
  - Run retrospectives to learn and improve
- Some organizations have standing Red Teams, continuously attacking the system in production: Microsoft, Intuit, Facebook, Yahoo, Salesforce

You can take many of the ideas from operations Game Days, intended to test the preparedness of the system and of the DevOps team under system failure scenarios, and apply them to InfoSec attack scenarios.

Do this by setting up two teams:

- Red Team: researches and designs attack scenarios and implements them (ethical hackers)
- Blue Team: identifies and responds to the attack, tries to contain the attack or close it off (the operations/incident response team)

Just like DevOps Game Days, security fire drills could be run in a test/simulation environment, but are most often done – and most valuable when done – in production. The goals of these exercises are to:

- Identify vulnerabilities in the system and weaknesses in controls that can be exploited (the job of the Red Team)
- Measure how quickly and easily the Red Team can exploit these weaknesses
- Verify that the Blue Team can identify an attack
- Measure how quickly and how effectively the Blue Team responds – and identify opportunities to improve their tools, playbooks. communications and escalation processes
- Identify gaps in testing and in design/implementation based on vulnerabilities and weaknesses that the Red Team exploits – and work on correcting them.

This approach is described in detail in the article: "Red Team versus Blue Team: How to Run an Effective Simulation", CSO Online, Mar. 25, 2008 http://www.csoonline.com/article/2122440/emergency-preparedness/red-team-versus-blue-team--how-to-run-an-effective-simulation.html

"Red Team: Pwning the Hearts and Minds one Ticket at a Time"
http://www.devsecops.org/blog/2015/12/10/red-team-pwning-the-hearts-and-minds-one-ticket-at-a-time

The book "Red Team: How to Succeed By Thinking Like the Enemy" by Micah Zenko, explores how Red Teams can be effective in many different scenarios, including and outside of security
http://www.amazon.com/Red-Team-Succeed-Thinking-Enemy/dp/0465048943

Security War Games at Microsoft

Microsoft, for example, conducts regular double-blind Red/Blue team exercises in production for their online services like Visual Studio Team Services. The rules of engagement for these exercises are simple: the Red Team cannot compromise customer SLAs, cannot DOS the system and cannot destroy or exfiltrate data. Common attack scenarios include server compromise, inside attacks, malware outbreaks, attempted DOS attacks. The Red Team measures MTTC/MTTP (Mean Time to Compromise/Mean Time to Pwnage), while the Blue Team measures MTTD/MTTR (Mean Time to Detect and Mean Time to Recover). The goal of the exercises is to test and exercise the incident response and forensics capabilities of the Blue Team, assess the tools used, and make the Blue Team stronger. At the end of each exercise, they run in-depth retrospectives to learn and improve.

"Security War Games with Sam Guckenheimer at Rugged Devops RSAC 2016", https://soundcloud.com/owasp-podcast/security-war-games-with-sam-guckenheimer-at-rugged-devops-rsac-2016

Red Team Exercises at Facebook

A fascinating look at Red Team exercises at Facebook: http://arstechnica.com/security/2013/02/at-facebook-zero-day-exploits-backdoor-code-bring-war-games-drill-to-life/

"Red Teams: When you can't find the bad guys, make some up" https://medium.com/starting-up-security/red-teams-6faa8d95f602

Red Team Hacking at Yahoo

"How Yahoo Hacks Itself" http://www.bankinfosecurity.com/interviews/how-yahoo-hacks-itself-i-3073

Case Study: Red Team Fire Drills at Intuit

- Intuit's security team runs Red Team exercises every week
  - Red Team scans for vulnerabilities, identifies targets, and creates attack plans
  - Targets are published each Friday internally
  - Red Team exercises are conducted on Monday
  - Report Cards are published on Tuesday – what was found, what needs to be fixed
- Provides constant feedback to developers and operations on vulnerabilities
- Teams often work on weekends to fix up problems in targets before the exercise starts

Red Team exercises are an important part of the security program at Intuit. A small offensive team conducts Red Team exercises every week. They start by identifying potential weaknesses through reviews and scanning and identify a set of target systems to be attacked. They publish this list within engineering each Friday. On Monday, they run the exercises through – in production. The Red Team can do anything except for take down a production service. Findings are published each Tuesday to engineering teams and managers.

Continuously hacking themselves has helped Intuit not only improve the security of their own systems, but the security of their cloud service providers as well. Providing real and frequent feedback to engineers encourages them to fix problems quickly – and to change how they work in order to prevent problems from being caught. In many cases, teams whose services have been identified as targets on Fridays will work over the weekend to try to find and fix problems in their code on their own, so that their scorecard will be better the following week.

"#RedTeamMonday" http://www.devsecops.org/blog/2015/11/11/redteammonday

"Intuit's DevSecOps: War Games, Gamification, and Culture Hacking"
https://www.voxxed.com/blog/2016/03/intuits-devsecops-war-games-gamification-culture-hacking/

Lab #2.3 – Monitor with Dashboards

Estimated Time: 15 Minutes

Use grafana and statsd to create dashboards showing some application metrics.

OBJECTIVES

Create a simple dashboard using grafana.

Add instrumentation to the Bricks application using statsd.

PREPARATION

Have your VM running and Bricks deployed

This page intentionally left blank.

Lab #2.3 Summary – Monitor with Dashboards

In this lab, you ...

- Created a new graph, based on an existing one
- Added security instrumentation to the Bricks application
- Created a new dashboard to monitor that instrumentation
- Exploited an SQL Injection flaw in the Bricks application

Having built a dashboard by starting from an existing example, we see how simple it is to add one more view of the data which is being collected. Additionally, after adding measurements for failed and successful logins, we can see how easy it is to start collecting new data. Together, these two simple skills can greatly increase our ability to understand how our applications are behaving.

Secure DevOps in Production – Summary

- ✓ Post-deployment smoke tests and checks to catch regressions and configuration mistakes
- ✓ Monitor security errors and attacks in application monitoring
- ✓ Run Game Days/Red Team exercises to exercise security incident response
- ✓ Consider using RASP, advanced WAFs or other runtime defense solutions to add visibility and runtime protection for legacy apps (especially Java and .NET)

This page intentionally left blank.

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- *Managing Secrets in CD*
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.

What Are Secrets?

Every system has secrets that need to be kept safe:

- OS and application passwords
- SSH keys
- TLS certificates/keys
- GPG keys
- API tokens
- Database credentials
- Sensitive runtime variables...

Automated Configuration Management and tooling in CD requires secrets to make changes

Every system has secrets: the credentials used to set up and configure and access the infrastructure, application services and the application itself. Automated tooling in CI/CD, automated configuration management tools, containers… all require secrets to access services or make changes.

As organizations automate provisioning, configuration management, deployment and release activities, implementing more tools and cloud services which require privileged access and credentials, the problems of how to manage secrets safely are becoming more important. It is too easy for people to make naïve mistakes and expose their organizations to serious risks.

How Not to Keep Secrets... In Code

Many engineers naturally think of keeping secrets in code as they try to automate builds or operations work

Storing secrets in code is a common anti-pattern

- This means that secrets are readable by anyone with access to the code repo
- Repo version history ensures that even if you remove a secret from the code, it can still be found
- In a distributed VCS like Git, everybody has a copy of the repo (including on their laptops) – you can't track who has it
- You must redeploy code every time that you rotate a secret

Secrets have to be kept secret. Kept out of plain-text configuration files, scripts, code, command-line variables… They need to be tracked and managed in an auditable and traceable way and easily rotated in the event of compromise (or to reduce the chance of compromise).

A common anti-pattern for managing secrets is to store passwords, keys and other credentials in code, and check them into the source repository.

This effectively gives the secrets away to anyone who has access to the code repository. And if the code is kept in a public repo, for example in GitHub, this could mean anyone, anywhere…

Even if passwords are stored in an encrypted form in a private source registry, keeping secrets in code means that you have to deploy new code any time that you want to change or rotate secrets.

Be Especially Careful with GitHub

It is too easy for someone to accidentally upload code with secrets to a public GitHub repo – especially if your team is contributing back to Open Source projects

- Contributing code such as Slackbot examples containing private Slack API tokens by accident, allowing account takeovers
- Uber was attacked using a database key which was accidentally uploaded to a public repo

Find secrets and sensitive data before attackers do!

- Regularly scan GitHub to check for sensitive information in public repos using: GitRob or Truffle Hog

There have been high-profile problems with people accidentally storing secrets in public repos on GitHub

"Accidental API Key Exposure is a Major Problem" https://rosspenman.com/api-key-exposure

http://arstechnica.com/security/2013/01/psa-dont-upload-your-important-passwords-to-github/

This includes people accidentally uploading Slackbot API tokens with example code: information that could be used to compromise Slack sessions. http://www.infoworld.com/article/3064355/security/how-you-might-be-leaking-your-secrets-onto-github.html

Uber is one of the most high-profile examples. Someone at Uber accidentally uploaded code containing a database key which was used to compromise information in Uber's driver database http://arstechnica.com/security/2015/03/in-major-goof-uber-stored-sensitive-database-key-on-public-github-page/

See also: "Users Scramble as GitHub Search Exposes Passwords, Security Details" http://www.webmonkey.com/2013/01/users-scramble-as-github-search-exposes-passwords-security-details/

It is relatively easy for attackers to find sensitive information using GitHub's code search features. http://www.securityweek.com/github-search-makes-easy-discovery-encryption-keys-passwords-source-code

You have to find this data before attackers do. Use one of the following tools

GitRob (https://github.com/michenriksen/gitrob)

GitRob is an Open Source tool that you can use to scan GitHub repos for files containing sensitive information It will catalog all repos related to an organization and scan for a list of different kinds of files that could be

dangerous. It will also identify information such as IP addresses, domains, tokens and email addresses in code
To review the specific signatures that GitRob checks, see:
https://github.com/michenriksen/gitrob/blob/master/signatures.json

For more information from the author:

- http://michenriksen.com/blog/gitrob-putting-the-open-source-in-osint/
- http://michenriksen.com/blog/new-version-of-gitrob-is-out/

TruffleHog (https://github.com/dxa4481/truffleHog)

Truffle Hog does deep searching of GitHub repos to find high-entropy strings. It searches the entire commit history of each branch.

Preventing Secrets in Code

Look for secrets in code reviews

Scan for secrets in code as part of commit checks...

- Custom grep searches
- FindSecurityBugs plugin...

or, better...

Prevent secrets from being committed to code repos using pre-commit hooks or plugins – easy to do with Git using these tools:

- git-secrets
- Talisman
- Git Hound

To prevent people from checking secrets into the code repository, you will need to ensure that reviewers look for secrets in code reviews.

There are also static checking tools that can help with this. For example, FindBugs and especially FindSecurityBugs have built-in checking for hard coded passwords and keys.

https://www.ysofters.com/2015/07/01/extending-findbugs-to-detect-hard-coded-passwords/

https://www.ysofters.com/2015/08/06/detection-of-hard-coded-passwords-improved/

Or you can write custom grep searches for secrets in source code.

There are also tools to prevent engineers from checking code in that contains secrets:

git-secrets (https://github.com/awslabs/git-secrets/blob/master/README.rst) is a tool that allows you to prevent secrets from being committed to Git repositories.

Talisman, from ThoughtWorks (https://github.com/thoughtworks/talisman). A tool is to validate code changes that are to be pushed out of a local Git repository on a developer's workstation. By hooking into the pre-push hook provided by Git, it validates the outgoing changeset for things that look suspicious - such as potential SSH keys, authorization tokens, private keys, etc.

Git Hound (https://github.com/ezekg/git-hound) is a Git plugin that prevents sensitive data from being committed to GitHub. It looks for matches to regular expressions specified in a config file and can stop the commit from proceeding. It can "sniff changes since last commit and pass to git-commit when clean"

http://www.infoworld.com/article/3155904/security/git-hound-truffle-hog-root-out-github-leaks.html

Handling Secrets in Configuration Management Tools

CM Toolset	Secrets Keeper	Description
Puppet	Blackbox	Store secrets in encrypted form in version control repo, created by StackExchange Originally designed to support Puppet but can be used in other ways
	Hiera-Eyaml	Asymmetric encryption of data in Puppet's Hiera backend store
Chef	Encrypted Data Bags	Data store for Chef (key value pairs) – data can be encrypted using RSA asymmetric keypairs
	ChefVault	OSS tool originally developed by Nordstrom, contributed to Chef – makes it easy to encrypt data bags using the public keys of a list of chef nodes
Ansible	Ansible Vault	Feature in Ansible to encrypt sensitive data which can be distributed or checked in to source control

Managing infrastructure through code using tools like Puppet or Chef doesn't make the problem of managing secrets go away. In fact, it adds to the problem. You still need to protect the secrets used by these tools, as well as protecting the key used to protect the secrets (creating a "bootstrapping problem").

Handling Secrets in Puppet

A good overview of secrets management with Puppet can be found here: https://puppet.com/blog/using-node-side-secrets-puppet

You must not store passwords in Puppet manifests – otherwise, anyone with read access to the repo where you hold the manifest code will be able to read the passwords.

One way to get around this is by storing passwords in Hiera (a backend key/value store for Puppet configuration data). However, the data for Hiera has to be stored somewhere – another repo. A tool called hiera-eyaml (https://github.com/TomPoulton/hiera-eyaml) provides asymmetric encryption of data in Hiera. This means that the public key can be given to developers who need to set secrets, but only the Puppet Master/Server can read them.

As Rafal Rzepecki explains, this makes the Puppet Master into a keymaster:

> "... It literally holds the keys to the (metaphorical) kingdom. This would be fine for a machine dedicated as that, but as it is, every administrator of the Puppet master can now surreptitiously copy the private key and then peruse the secrets from source control at their leisure, unbeknownst to anyone. Ditto for an attacker who would somehow compromise the machine, only if just once, and just for a moment."

Another problem is that this approach only works if Puppet is the only tool that needs to access and work with secrets.

Rzepecki recommends instead that each node obtains secrets as they need them, from a back-end secrets server. This can be done for example using Summon (http://conjurinc.github.io/summon/), an open source command-line tool which retrieves passwords from different secrets servers (including Conjur, which open sourced the Summon tool).

SSH key issues with Puppet

https://www.phase2technology.com/the-art-of-ssh-key-management-with-puppet/

https://ask.puppet.com/question/476/securely-storing-passwords-and-keys/

http://stackoverflow.com/questions/11171472/handling-sensitive-information-with-puppet

https://groups.google.com/forum/#!topic/puppet-users/uC3_Uo8vqeQ

BlackBox (https://github.com/StackExchange/blackbox)

BlackBox allows you to store secrets and credentials in a version management system (Git, Mercurial, svn, perforce) in encrypted form. Does not require retrieval of secrets by Puppet or other software, only decryption prior to deployment.

"The BlackBox project", Tom Limoncelli, http://www.slideshare.net/TomLimoncelli/the-blackbox-project-sfae

Chef Vault (https://github.com/chef/chef-vault)

Open source tool to "encrypt a Chef Data Bag Item using the public keys of a list of chef nodes. This allows only those chef nodes to decrypt the encrypted values." Chef Vault was originally developed as an open source project by Nordstrom, and transferred to Chef in 2015.

https://docs.chef.io/chef_vault.html

https://www.chef.io/blog/2016/01/21/chef-vault-what-is-it-and-what-can-it-do-for-you/

https://github.com/chef-cookbooks/chef-vault

Using Hashicorp Vault with Chef: https://www.hashicorp.com/blog/using-hashicorp-vault-with-chef.html

Ansible Vault (http://docs.ansible.com/ansible/playbooks_vault.html)

Feature in Ansible (1.5) for keeping sensitive data such as passwords or keys in encrypted files – the vault files can then be distributed or placed in source control.

"Working with Ansible-vault" https://gist.github.com/tristanfisher/e5a306144a637dc739e7

https://dantehranian.wordpress.com/2015/07/24/managing-secrets-with-ansible-vault-the-missing-guide-part-1-of-2/

https://dantehranian.wordpress.com/2015/07/24/managing-secrets-with-ansible-vault-the-missing-guide-part-2-of-2/

Secret Keepers

Secrets Servers or Secret Keepers securely store, manage and distribute secrets to users, tools, and servers

- Encrypted storage of credentials, keys etc.
- API and CLI access for tools like Chef, Docker, and Puppet and for applications
- Secrets management: functions to enroll, revoke, rotate
- Auditing of setup and changes to secrets, and use of secrets
- Authentication and access control filtering

How to choose a Secret Keeper:

- Platform support: there are several secrets management solutions which are only available on AWS.
- API support – how open and accessible is the service to other tools and applications?
- Enterprise support: clustering, HA, reporting capabilities – if your apps and other services are going to rely on this service for secrets, it has to be reliable
- Secrets management capabilities: enrollment, revocation, rotation all need to be supported
- Strength of security – how well protected are the keys and passwords, and how confident are you that the solution is itself secure and safe? Do a careful threat assessment of the solution first
- Support: how active is the community (OSS) or how effective is the support offering from the vendor?

David Somerfield's presentation at OWASP AppSec USA 2015 on approaches to managing secrets:

"Turtles all the way down: Storing Secrets in the Cloud and in the Data Center"

https://www.youtube.com/watch?v=OUSvv2maMYI&feature=youtu.be

For an excellent overview of available secret managers, see:

https://gist.github.com/maxvt/bb49a6c7243163b8120625fc8ae3f3cd

Secret Keepers – Open Source Solutions

Tool	Description
Vault	Hashicorp OSS – and enterprise version, designed to be highly extensible Probably the most stable and mature of these offerings, and the most popular Supports multiple back ends (including AWS) and data stores Enterprise version includes an admin interface, status dashboard, a well-defined workflow for initialization and unseal, and HSM integration
Keywhiz	Square OSS – provides access through REST API or FUSE filesystem
Sneaker	AWS only, store secrets on S3 using Amazon KMS
Confidant	From Lyft – for AWS, stores secrets in DynamoDB
Knox	Pinterest OSS designed to be easy for developers to use and simple to rotate secrets

There is a choice of open source Secrets Servers available, including:

Vault from Hashicorp (https://www.vaultproject.io/)

Vault is a secrets server written in Golang, by Hashicorp, the company behind popular open source tools such as Vagrant and Consul. It provides central storage and management of secrets, with HA capability. Secrets are safely stored: all data is encrypted using AES-GCM-256, including the master key used to encrypt the data. The master key is split using Shamir's Secret Sharing (https://en.wikipedia.org/wiki/Shamir's_Secret_Sharing), At least 3 keys (configurable) are required to unseal the vault. Access to the vault is through TLS 1.2. Vault has undergone extensive 3rd party security reviews.

Every secret in Vault has a lease (a validity term), which must be renewed periodically through the API. There are also API functions to revoke and rotate secrets. Vault also provides "dynamic secrets": temporary credentials which are created on the fly and automatically expired/deleted. For example, any application that requires SQL database access can automatically generate a user with a temporary password which will automatically be deleted.

Vault is probably the most stable and mature of the open source secret keepers.

https://hashicorp.com/blog/vault.html

https://www.codementor.io/devops/tutorial/how-to-install-vault-hashicorp-secure-deployment-secrets

Keywhiz from Square (https://square.github.io/keywhiz/)

Keywhiz is another open source secrets manager, from Square based on technology that Square uses internally. It provides access to secrets through a RESTful API or through a FUSE filesystem, using mutually authenticated TLS (mTLS). Keywhiz handles complex environments where different systems need different secrets deployed. As of January 2016, Keywhiz is still in alpha testing.

Sneaker (https://github.com/codahale/sneaker)

Sneaker is a tool for securely storing secrets on S3 using Amazon Key Management System (KMS). Secrets are stored on S3, encrypted with AES-256-GCM and single-use, KMS-generated data keys.

Confidant (https://github.com/lyft/confidant)

An open source tool from Lyft for AWS users, which stores secrets in encrypted form in DynamoDB.

https://eng.lyft.com/announcing-confidant-an-open-source-secret-management-service-from-lyft-1e256fe628a3

Knox (https://github.com/pinterest/knox)

Open source secrets keeper from Pinterest in Go (open sourced May 2016). Designed to be simple for developers to use, and to make it easy to rotate keys.

https://engineering.pinterest.com/blog/open-sourcing-knox-secret-key-management-service

https://github.com/pinterest/knox/wiki

Secret Keepers – Commercial

Tool	Description
Amazon KMS	Managed Amazon service to create and control encryption keys Uses HSMs to encrypt keys Integrates with CloudTrail to audit key management and usage
Conjur	Expensive enterprise secrets management solution, also provides ssh management functions Highly Scalable and distributed Secrets Management Vault is Part of a larger "Trust Management Platform"
Thycotic	Centralized secure password manager
Password Manager Pro	Simple secrets manager/database, not scalable More suitable for storing individual credentials

Amazon KMS (https://aws.amazon.com/kms/) is a managed service from Amazon that allows you to create and manage (create, import, rotate and revoke) encryption keys for other services and for your application. It uses Hardware Security Modules (HSMs) to encrypt keys. KMS integrates with other AWS services and supports auditing through AWS CloudTrail. Includes a SDK for application developers.

Conjur (https://www.conjur.net/) is an expensive enterprise secrets server that also provides ssh management functions. Conjur's Secrets Management Vault is part of its larger "Trust Management Platform"

Thycotic (http://thycotic.com/products/secret-server/) is a centralized secure password manager

Password Manager Pro (https://www.manageengine.com/products/passwordmanagerpro/) is a simpler password management solution that is good for API access from a fixed number of instances, but does not scale easily (every client must be manually configured)

Secret Keepers – Using Vault

Preparing Vault for use requires some setup work:

- Configure storage backend, TLS certificates, telemetry
- Launch and Initialize Vault – Securely distribute unseal keys
- Unseal the Vault – Requires a quorum of unseal keys
- Enable authentication backend(s) – Create users if needed
- Define access control policies
- Generate & distribute initial tokens

- **Revoke root token** – Regenerate via quorum of unseal keys

Setting up vault to provide a secure service where secrets can be stored takes care.

The initial configuration of Vault covers only the items necessary to access the service over the network, to store the secrets data properly, to monitor the service, and to define default behaviors. Additional secrets backends can be added to the configuration later, if needed. For the lab, the "file" back-end is used for simplicity sake. Other options include etcd, consul, inmem, s3, azure, zookeeper, and several database engines.

Once Vault is configured and running, it needs to be initialized and unsealed. The initialization process creates an initial root token and a set of unseal keys using Shamir's Secret Sharing algorithm to split the keys in a manner which requires a quorum to unseal the vault. For example, you may choose to initialize the vault with 10 key shares and require 3 to unseal the vault. Should the vault service stop, 3 of the 10 people with key shares will need to collaborate to restore the service. This makes vault resistant to automating the service restart; this is by design, and helps protect the data inside the vault. Even the open-source version of Vault supports high-availability configurations to help reduce the likelihood of needing a service restart unexpectedly.

After the Vault is unsealed, authentication backends can be configured. While this may seem out-of-order, it allows the configuration for authentication services to be stored securely *inside* the vault, rather than being kept in a configuration file outside the vault. All Vault authentication backends operate by validating an asserted identity and then generating a token which is then used to access the vault service. By default, only the 'token' authentication back-end is enabled.

In addition to authentication, Vault has a robust authorization engine as well. After the vault is unsealed, access control policies should be created to limit the scope of access that users' tokens have. The built-in 'root' access policy gives any process/user with the root token the ability to do *anything* in vault. Naturally, access to the root token should be carefully controlled.

Having access control policies in place, we can now generate initial tokens to use in our applications. In many cases, this is not strictly required, as a user or process can authenticate using another authentication back-end and get an access token on-the-fly.

At this point, the vault environment is ready for use. Because of the inherent risks of keeping a root token active in the system, the recommended practice is to revoke the initial root token once the configuration tasks above have been completed. Should a root token be needed in the future, it can be regenerated by a quorum of unseal key holders.

https://www.vaultproject.io/docs/concepts/index.html

https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

Secret Keepers – Vault Configuration

For our lab environment, Vault is configured with:

- Secrets stored in the "file" back-end
- Authentication via the "token" back-end, only
- TLS configured, and telemetry using statsd
- 200-day leases on tokens (not recommended)
- 5 unseal key shares, 2 required for quorum (set at initialization)
- Policies for Jenkins, Bricks, and the student user
- Keys created for Jenkins, Bricks, and Student

Here's the configuration for vault used in our lab VM:

```
backend "file" {
  path = "/vault/file"
}
listener "tcp" {
  address = "0.0.0.0:8200"
  tls_cert_file = "/vault/config/vault.sansappsec.org.crt"
  tls_key_file = "/vault/config/vault.sansappsec.org.key"
  // The Jenkins vault plugin only supports TLS 1.0  :-(
  tls_min_version = "tls10"
}
telemetry {
  statsd_address = "graphite.sansappsec.org:8125"
  disable_hostname = false
}
cluster_name     = "sansappsec"
default_lease_ttl = "4800h"
daysmax_lease_ttl    = "9600h"
```

The long token leases are needed to ensure that the token will be valid for the expected lifetime of the lab VM. Key shares are set when the vault is initialized. Policies and keys were set after the vault was unsealed.

Lab #2.4 – Protecting Secrets with Vault

Estimated Time: 15 Minutes

Use vault to store the database password for the Bricks application.

OBJECTIVES

Use "git grep" to find secrets in the Bricks code.

Incorporate vault support into Bricks application and confirm usage

PREPARATION

Have your VM running and Bricks deployed

This page intentionally left blank.

Lab #2.4 Summary – Protecting Secrets with Vault

In this lab, you ...

- Used git grep to search for passwords in the Bricks application
- Used GitLab's merge request capability to review code
- Modified Bricks to use Vault to gather the database password
- Used the Vault command line to confirm that Bricks is using vault

This page intentionally left blank.

Managing Secrets – Summary

- ✓ Watch out for engineers taking shortcuts in managing secrets
  - Don't allow secrets in source repos or plain-text files or runtime variables
- ✓ Use a secret keeper/vault to safely store and distribute secrets – access it through APIs
- ✓ Plan to expire and rotate secrets – do this using your CD pipeline to make sure that the change is automated, audited, and traceable

This page intentionally left blank.

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.

DevOps and Compliance

What do you need to prove to an auditor?

- Awareness and understanding of risks (operational, technical, security, privacy, legal) and that they are being managed effectively – at all levels
- Demonstrate Change Control approvals
- Testing and reviews are always done consistently
- Separation of duties is enforced
- Vulnerabilities are managed responsibly
- Corrective actions: failure of controls are understood and fixed
- Traceability of all steps

We will look at how to do all of this using automated tooling and CI/CD workflows

"Compliance as Code" is an approach popularized by Justin Arbuckle at Chef, where compliance policies are documented and enforced automatically through code:

- Controls and policies are agreed to and defined in configuration management code like Chef cookbooks or Puppet manifests – using standard benchmarks like CIS where available
- Reviews and automated tests ensure that policies are coded correctly
- All changes are made through the automated CD pipeline – transparent and consistent
- Online detective change control/drift analysis alerts when policies are being violated
- All of these activities are logged for assurance purposes

Towards Compliance as Code
https://dzone.com/articles/towards-compliance-code
https://www.chef.io/blog/2015/05/11/towards-compliance-as-code-a-real-world-example/
http://devops.com/2014/11/04/automating-away-regulatory-compliance-myth/

Successfully Establishing and Representing DevOps in an Audit, James Deluccia, FlowCon
https://www.youtube.com/watch?v=PijCUJDb_hc

Splitting the Check on Compliance and Security: Keeping Developers and Auditors Happy in the Cloud, Jason Chan, Netflix, AWS re:Invent, October 2015 https://www.youtube.com/watch?v=Io00_K4v12Y

Keeping the Auditor Away: DevOps Audit Compliance Case Studies: Gene Kim and James Deluccia
http://www.slideshare.net/realgenekim/keeping-the-auditor-away

Preparing for a SOC2 Audit with DevOps in Mind, Brian Henerey, DevOpsDays Chicago 2014
https://www.youtube.com/watch?v=G6Q9TGcNWqg

DevOps Audit Defense Toolkit

Process framework for secure Continuous Deployment and operations

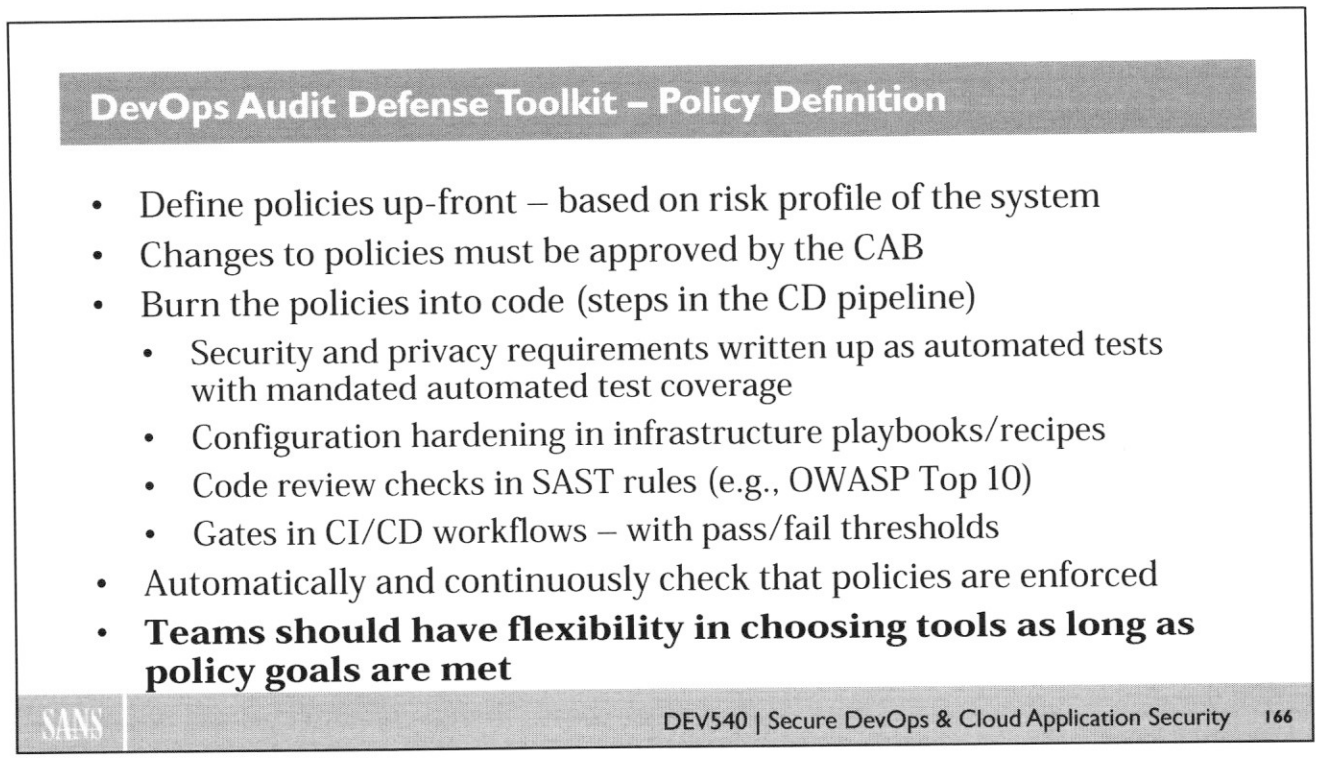
- Understand risks and define control program upfront
- Covers application and infrastructure changes
- Leverages CD workflow steps and tools
- Traceability is provided through electronic tickets and version control, and built-in activity logs
- Relies on highly-disciplined and consistent engineering practices
- Controls/evidence provided must be audited periodically

The DevOps Audit Defense Toolkit (http://itrevolution.com/devops-and-auditors-the-devops-audit-defense-toolkit/) is a free community-built resource demonstrating how compliance controls and checks can be automated and built into DevOps workflows. It was written by experts in IT compliance and DevOps and implements current best practice in Agile and Continuous Delivery/Continuous Deployment. The principal authors are James DeLuccia IV, Jeff Gallimore, Gene Kim and Byron Miller.

The Toolkit follows a case study format for a fictional organization, outlining a highly-disciplined set of practices and workflows for managing and deploying changes to code and infrastructure. It builds on Continuous Delivery and Continuous Deployment, Infrastructure as Code, secure DevOps practices and examples from DevOps organizations working in regulated environments.

The control environment is based on a consistent set of development and operations practices, automated as much as possible. You will need to periodically audit to ensure that these practices are being followed. It leverages Continuous Integration/Delivery/Deployment automation to ensure consistency and provide an audit trail of changes.

You can participate in the conversation and contribute back to the Toolkit at https://plus.google.com/communities/103372669680429508474



Policies need to be defined by management, compliance, development, ops, the PMO and InfoSec working together. Management needs to be confident that operational and technical and security risks and compliance and governance reporting requirements will be satisfied with the appropriate checks in CI/CD.

Any changes to policies, rules or the workflow definition should be reviewed and approved, for example in a CAB meeting.

These policies are then implemented in code:

- Chef cookbooks or Puppet manifests which follow hardening guidelines or standards such as CIS
- Automated checks in the CD pipeline. This can include: ensuring that code reviews are done pre-commit, SAST and DAST scans (with minimum bars which will fail the build), automated tests with test coverage requirements, supply chain verification (checking dependencies for vulnerable components), automated attacks using Gauntlt or similar tools
- Automated asserts/checks in test, staging, and production – following the model of Netflix's Security Monkey…
- Automated deployment – runbooks and checklists scripted or coded and executed consistently
- Continuous automated detective change control/drift analysis to alert on deviations from baseline

DevOps Audit Defense Toolkit – Change Control

How do you enforce change control when developers are pushing changes out directly in Continuous Deployment?

- "Conditions of Satisfaction" are defined up front for each change with the Product Owner, and must be met before it can be closed
- Tests are written to prove that these conditions are met
- Peer reviews check that each change meets these conditions
- When the change is "done", it has been accepted – ready to go to production
- Most changes are small and incremental – minimal risk, can be treated as standard/routine changes (do not require CAB review)
- Data/database changes and larger upgrades need more planning and reviews, but should leverage the same workflow/controls and tools

A challenge with Continuous Deployment – developers pushing changes directly to production – is how to enforce change control and change management approvals.

In Agile and DevOps environments, change control and approval is done in phase on each change as part of the team's development workflow, rather than waiting until the end of the project. Before stories can be added to a Sprint backlog or changes are added to a Kanban board, they must have "Conditions of Satisfaction" agreed to (https://www.mountaingoatsoftware.com/blog/clarifying-the-relationship-between-definition-of-done-and-conditions-of-sa). These are a set of concrete, testable conditions that must pass for the feature/change/fix to be accepted by the team's Product Owner or Product Manager. Peer reviewers check to ensure that the conditions are met, by looking at the code and the tests.

Once a change is "done" (coded, tested, reviewed, and demoed to the Product Owner), this means that it has been accepted and approved by management, and is ready to be deployed. Because most changes in Agile and DevOps are incremental, they will be small and should be low risk. Therefore, they can be treated as routine or standard changes that do not require separate review and approval by a CAB.

The Audit Defense Toolkit expects that developers will test their own work, and that acceptance testing is not necessarily required as a separate step. In this model, all changes that pass through the automated testing pipeline will be deployed directly to production in Continuous Deployment. However, additional reviews and acceptance testing can be added before changes are deployed into production in Continuous Delivery.

Larger changes that require more planning and reviews should be broken down into incremental steps where possible, and "launched dark", with the change or new feature disabled until an operational readiness review is completed.

DevOps Audit Defense Toolkit – Peer Reviews

Peer reviews are an important control in CD:

- Mandatory reviews support Separation of Duties (chain of trust)
- Reviews are risk-focused (correctness, security, insider threats/fraud)

Every change is reviewed pre-commit

- Reviewers must be qualified (experienced, trained in secure coding)
- Consistent code review guidelines (for understandability and safety)
- Reviewers should be selected at random (to prevent collusion)
- High-risk changes reviewed by SME
- Reviews should include code and tests
- Reviews are recorded to provide evidence (using Git, Gerrit, etc)

The Audit Toolkit controls depend heavily on peer reviews, especially code reviews. Reviews are not only a quality control and learning tool. Mandating that reviews are done, and selecting reviewers at random, discourages insiders from making unauthorized changes or intentionally introducing backdoors or malware.

Reviews need to be done for risk, not for style. Leverage IDE templates and SAST tools to enforce consistent code format and good coding practices. It is important to ensure that security considerations (checking for defensive coding practices, ensuring that security libraries and features are being used properly, and watching out for common security coding mistakes) are included in code reviews and code review checklists. Developers – whether they are writing code or reviewing somebody else's code – need training in secure coding.

High-risk code (security plumbing, code that deals with PII/credit card data, public network-facing APIs, etc.) requires independent security review from a security specialist or at minimum a senior engineer (SME/specialist review). This code can be identified through quick code scanning (automatically checking for dangerous functions like crypto calls, random number functions, etc.), DAST or SAST scans, manual code reviews, pen testing, and by encouraging developers to ask for a specialist review whenever they feel it is required.

Evidence that reviews are done needs to be kept as part of the audit trail of every change – this is easy if you are using Git, or an automated code review tool like Gerrit, Crucible or Review Board.

DevOps Audit Defense Toolkit – Infrastructure as Code

Configuration changes are made the same way as application code changes

- Changes are made in code, reviewed pre-commit
- Checked in to repos under version control
- High-risk changes require SME review
- Static analysis/lint checks in CI/CD
- Automated tests through rspec-puppet, Test Kitchen, or Serverspec
- Deployed to test and staging in sequence, using automated pipeline

Changes to system configuration are made following the same workflow and controls as application changes, using an automation tool like Puppet or Chef or Ansible and/or Docker, plugged in to the CI/CD pipeline. This includes

- Pre-commit code reviews
- Static analysis checking
- Automated testing.

Changes to infrastructure are promoted through test environments to staging and finally to production as long as all of the tests and checks pass.

This provides visibility into configuration, ensures consistency, and provides traceability of changes.

DevOps Audit Defense Toolkit – Traceability

Every step is audited

- Traceability through tickets which capture requirement and "Conditions of Satisfaction"
- Track back to the original change request or bug report – from check-in, to review, test, and deployment
- Take advantage of logs provided by tools (Git, ticket system, code review tools, CI/CD server, automated tests, scanning, ...)
- Logs need to be write protected and archived

All steps in the CD pipeline are timestamped and logged, from code check-in to testing and deployment.

Traceability is enforced through tickets in Jira or some other tracking system. Every change is tagged with the ticket (code check-ins, code reviews…). This provides traceability back to the original request: who asked for the change or fix, why, who approved it, who made the change and when and what tests were checked-in for the change when the change was tested and deployed.

All of this takes advantage of the logs which are automatically created by the tools involved. This means that logs need to be write protected and archived.

DevOps Audit Defense Toolkit – Separation of Duties

The Audit Toolkit helps to enforce/support Separation of Duties

- Mandatory peer reviews of all changes to code and infrastructure – reviewers are assigned randomly where possible
- Developers are allowed in production to troubleshoot failures – read-only access
- All changes to production code or configuration are made through the CD pipeline (roll forward or back)
- Detective change control is used to catch unauthorized changes
- Access restrictions enforced and regularly reviewed
- Production access logs are reviewed regularly

The principle of Separation (or Segregation) of Duties is intended to reduce the chances of fraud and security breaches by preventing unauthorized changes and minimizing the number of people who have access to sensitive data. It dictates that no individual should have end-to-end control over a change to code or data.

In the DevOps Audit Toolkit, a number of controls enforce or support Separation of Duties:

- Mandatory independent peer reviews ensure that no engineer (dev or ops) can make a change without someone else being aware and approving it – reviewers are assigned randomly to prevent collusion
- Developers are granted read-only access to production systems to assist with troubleshooting. Any fixes need to be made through the CD pipeline (fixing forward) or by automatically rolling changes back (again through the CD pipeline/automated deployment processes) which are fully auditable and traceable
- All changes made through the pipeline are transparent, published to dashboards, IRC, etc.
- Production access logs are reviewed by IT Operations management weekly – including correlation of developer access and code deployments
- Access credentials are reviewed regularly
- Automated detective change control tools (e.g. Tripwire, OSSec, UpGuard) are used to check for unauthorized changes

DevOps Audit Defense Toolkit: A Work in Progress

- The DevOps Audit Defense Toolkit is a work in progress – use it to build on, customize, and extend for your environment
- May be too heavyweight for some environments – cherry pick controls and workflows to start
- Assumes Continuous Deployment – can be modified for Continuous Delivery

The DevOps Audit Defense Toolkit describes how you can automate many of your regulatory controls. By automating these controls and checks as part of the CD pipeline, you ensure consistency, and you also make sure that all of these checks are exercised continuously. This should increase your confidence in the quality of compliance. It is still a work in progress – a base that you can build on, customize and extend.

The Toolkit's workflows are detailed and extensive, and may be too heavyweight for some environments – at least to start. You may want to start by cherry-picking key controls.

On the other hand, it also assumes a self-service Continuous Deployment approach, where individual changes to code or infrastructure are pushed to production as soon as the change passes through the automated checks and workflow. You may need to modify this to support Continuous Delivery, where changes are bundled up at the end of the pipeline and later scheduled for deployment to production.

For more on the Toolkit, read "Defending DevOps" http://www.csoonline.com/article/2365915/it-audit/defending-devops.html

Vulnerability Management in DevOps

Challenges

- Compliance mandates that you perform regular vulnerability scanning and penetration testing, and prove that you acted on the results
- Rapid rate of change in DevOps makes scan/pen test results out of date quickly
- Short CI/CD cycle times, long vulnerability scan times
- Reporting formats are often difficult to ingest by tools, not immediately actionable
- Automated vulnerability scanners lack knowledge about business impact of various application components, features

Vulnerability management needs to be integrated into DevOps workflows and feedback loops:

- Because of rapid changes in DevOps (especially Continuous Deployment environments), vulnerability scans and remediation must be done much more often.
- Vulnerability scans may need to be run out of band because scans can take a long time.
- Findings from scans must be fed back into the team's backlog for prioritization and resolution. This is harder than it should be, because results need to be taken out of reports, de-duplicated, and false positives filtered out before they can be added to the backlog.
- We will look at some tools to help with managing this.

Penetration Testing in DevOps

Must be done for major changes and to meet compliance obligations

Get results into tickets, not reports

Treat major findings like a production incident

- Understand how to reproduce and fix the problem - fix it now, and fix it right
- Post mortem, 5 Whys, etc.,
- Why didn't we catch this/prevent this?
- Address root cause(s)
- Feedback into continuous improvement. Raise the bar for tests and reviews, training, hiring...

In Continuous Delivery/Deployment, changes are being made all the time and released to production frequently. Penetration tests cannot be used to "certify" every release because often there is no idea of a "release" at all, just a continuous stream of changes and fixes to production.

Instead, penetration testing should be used as a risk assessment activity, to find holes and weaknesses in the development and delivery and operations controls. The focus should not just be on what security vulnerabilities are in the system—but how they got there, and what needs to be changed or improved to prevent these problems in the future.

Penetration testing isn't an effective control gate in DevOps, especially in Continuous Deployment. Tests are too expensive, take too long, and the results will go stale too quickly. But it can help in assessing the team's (and system's) security posture. Of course, you may still need to conduct pen testing in order to meet compliance requirements—for example, once a year or after any major changes as required by PCI DSS. Understand that in Continuous Delivery there may not be any major changes—just a stream of small incremental changes.

By the time that you do schedule a penetration test, the application may already be in production or will be almost immediately—so you need to take penetration tests seriously and act on them immediately. Treat findings as production issues. Swarm and fix the important problems quickly. Then conduct a blameless post-mortem to understand the root cause(s) of the problems. Use techniques like 5 Whys (https://en.wikipedia.org/wiki/5_Whys) to try to understand where the weaknesses are in your training, design, coding, reviews, testing, packaging, configuration, deployment, and monitoring practices—and what you need to do to get better as an organization.

Get the results back to developers, track them and get them fixed and the fixes in. A tool that can help with this is ThreadFix (http://threadfix.org). ThreadFix tracks automated scan results (SAST, DAST) as well as pen test results, correlates findings, and can generate tickets in a bug tracking system (e.g. Jira) for developers to work on.

Managing Vulnerabilities using ThreadFix

ThreadFix is an example of a tool that provides a central repository of vulnerabilities

- Collects vulnerability data from different scanners, de-duplicates and pushes to defect tracking systems/IDEs
- Triage and prioritize vulnerabilities
- Can be integrated into CI/CD pipelines
- Open Source and Enterprise editions—although, new features are only being offered in the Enterprise edition as of version 2.4 (July 2016)

ThreadFix (http://www.threadfix.org/) is a central repository of vulnerabilities, which aggregates and correlates vulnerabilities from different scanners (SAST and DAST tools and scanning services such as WhiteHat Sentinel, QualysGuard WAS, and Veracode), manual code reviews, and penetration test results. https://github.com/denimgroup/threadfix/wiki/Remote-Providers

It uses Hybrid Mapping to de-duplicate findings between different kinds of scanners (so that a vulnerability found by multiple scanners including both DAST and SAST will only be reported once). Dynamic scanners supported include: Arachni, OWASP ZAP, Skipfish, and w3af, as well as commercial scanners such as IBM AppScan, HP WebInspect, and Acunetix. Static analysis tools supported include: Brakeman, CAT.NET, Cppcheck, Clang, Findbugs, PMD, and commercial scanners such as IBM AppScan Source and HP Fortify.

ThreadFix also pulls in vulnerability information from OWASP Dependency Check and Sonatype Nexus CLM for vulnerabilities in dependencies. https://github.com/denimgroup/threadfix/wiki/Component-Lifecycle-Management-Tools

It provides a central dashboard view over the security state of your application portfolio. You can create different views (by application type, by team ownership, geography), track vulnerability status and trends. And you can set alert thresholds based on changes to the vulnerability profile of an application.

ThreadFix pushes vulnerability details into bug tracking systems like Jira to fit developer workflows, and there are IDE plugins which take data from ThreadFix to pinpoint lines of code which have vulnerabilities.

https://github.com/denimgroup/threadfix/wiki/Defect-Tracker-Guide

https://github.com/denimgroup/threadfix/wiki/IDE-Plugins

As of July 2016, new features are only being made available in the Enterprise Edition of Threadfix (from version 2.4 forward)

https://groups.google.com/forum/#!topic/threadfix/bn2nnoWYhlg

ThreadFix exposes a REST API with a CLI and client libraries in Java, Python, and Go, to make it easy to integrate into CI/CD pipelines.

https://github.com/denimgroup/threadfix/wiki/ThreadFix-REST-Interface

There is also a Jenkins Plug-in which uploads supported scan files from Jenkins to ThreadFix server

https://wiki.jenkins-ci.org/display/JENKINS/ThreadFix+Plugin

http://www.denimgroup.com/blog/denim_group/2014/06/automation-domination-integration.html

ThreadFix Scan Agent: wraps dynamic scanners such as Burp Suite, OWASP ZAP, IBM Appscan, HP WebInspect…) and can control and coordinate runs, and feed the results back to ThreadFix. This is only available in the Enterprise Edition.

Open Source Community Edition

https://github.com/denimgroup/threadfix/

Commercial Enterprise Edition

Includes authentication and authorization controls, proxy support, and scan orchestration

http://www.threadfix.org/pricing/

PCI DSS and DevOps - Example

PCI DSS is a prescriptive and highly specific regulatory framework which can create challenges for DevOps teams

PCI strictly enforces production/development separation

- Separation of duties and restrictions on access to production systems
- Additional auditing requirements
- Separate push step – Continuous Delivery by Ops not Continuous Deployment by Dev

Netflix and Etsy are examples of DevOps shops that are PCI compliant

Many of these controls are described in the Audit Defense Toolkit

PCI DSS at Netflix

Splitting the Check on Compliance and Security: Keeping Developers and Auditors Happy in the Cloud, Jason Chan, Netflix, AWS re:Invent, October 2015 https://www.youtube.com/watch?v=Io00_K4v12Y

In this presentation, Jason Chan at Netflix maps PCI DSS (and SOX) requirements to DevOps controls and shows how disciplined Continuous Delivery/Deployment practices can meet compliance requirements. He lays out a few important points:

- A Continuous Delivery/Deployment pipeline can have – and may require – manual test and review steps as part of the CD pipeline
- Engineers and auditors both need to know: when did this code change? Who changed it – and who tested it? Did the tests pass? Who deployed it – and when? This kind of traceability from check-in through code review, testing and then to deployment comes as part of disciplined CD
- A microservices-based architecture makes it easy to isolate critical services that manage sensitive data such as token services and crypto proxies
- Automated deployment and continuous security visibility through tools like Security Monkey minimize (and audit) mistakes
- Build tools that help engineers, the security team and auditors!

PCI DSS at Etsy

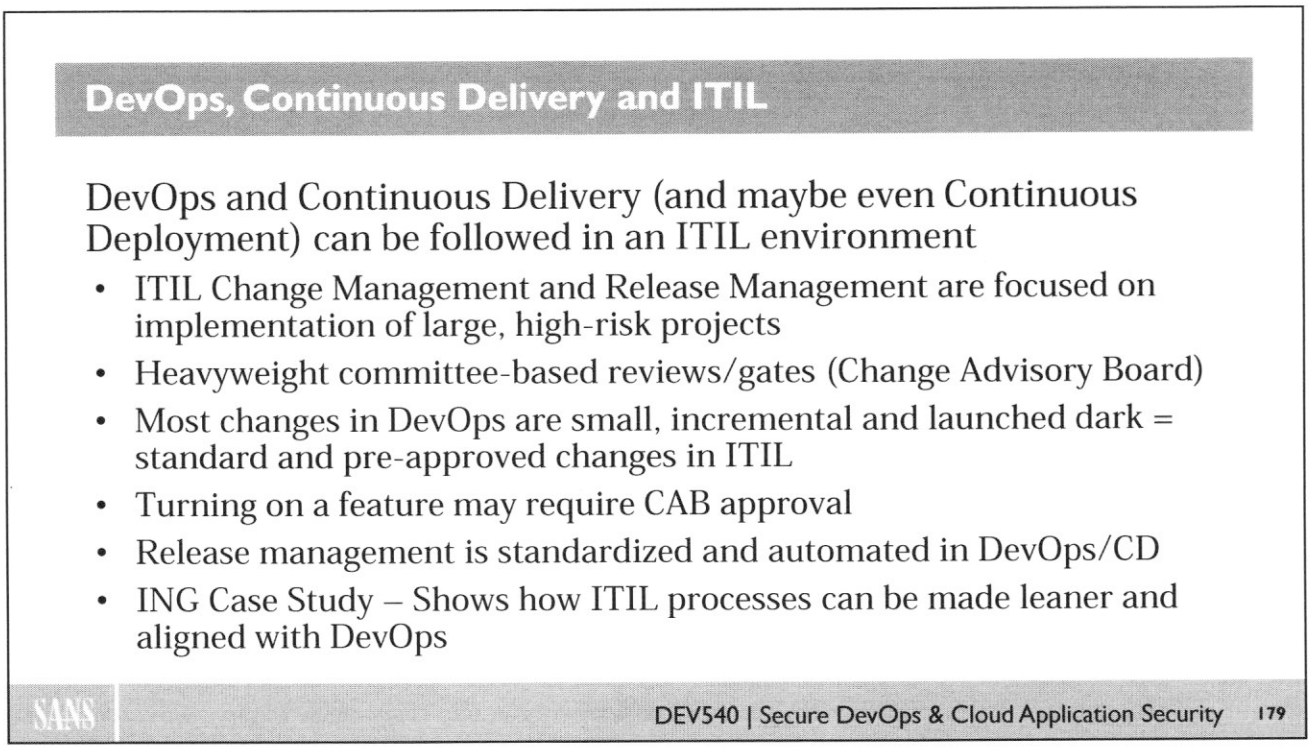
Etsy deploys to production 25-50 times a day. They built a new, segmented PCI-DSS compliant environment for their payment systems – "we built a whole separate Etsy, essentially"

Segregation of duties doesn't mean people can't talk to each other. In fact, collaboration is an essential element of effective risk management. So, everybody in the PCI environment still follows DevOps principles – there's no physical separation of the teams, and Etsy have hired more people into the various roles (dev, sysadmin, dba, manager, networking) to facilitate collaboration;

- In the payments environment, they "still have to follow the rules: a developer still doesn't have access to a production database", but they'll have DBAs working alongside them who they can ask for help, and graphs showing metrics from the database;
- They use a similar deployment process in their PCI environment to their non-PCI environment, but it includes much more logging on who did what when, and they have separate roles for QA pusher and production pusher;
- Developers can run most of the test framework in their PCI development environment, and they have access to production logs via Splunk in read-only mode;
- The PCI team has a ticketing system that all changes have to go through, but they can push out a change from dev to production in less than an hour if necessary with their normal, non-emergency change management process;

http://continuousdelivery.com/2012/07/pci-dss-and-continuous-deployment-at-etsy/

http://www.cio.com/article/2397663/developer/continuous-deployment-done-in-unique-fashion-at-etsy-com.html



ITIL is a well-defined, prescriptive standard for managing IT service delivery and operations. DevOps can be aligned with ITIL Configuration Management, Change Management and Release Management controls.

As discussed in the DevOps Audit Defense Toolkit, governance policies which define standard and normal changes and what approvals are required for each type of change, must be defined upfront.

Most changes in CD are small, incremental, and launched dark in multiple iterations. These changes could be (and should be) treated as standard changes, which are pre-approved. But turning on a feature could be considered a material change requiring CAB approval and coordination.

For more on change management considerations in DevOps see: "Continuous Delivery and ITIL: Change Management"
http://continuousdelivery.com/2010/11/continuous-delivery-and-itil-change-management/

Case Study at National Broadband Network
https://puppet.com/blog/a-deployment-pipeline-for-infrastructure
"ITIL vs DevOps": http://www.infoq.com/news/2015/06/itil-vs-devops

"ING ITIL and DevOps at War in the Enterprise"
https://www.youtube.com/watch?v=_dDsdbkSgOc
http://www.infoq.com/news/2014/06/ing-transtition-to-devops

"ITIL and DevOps can be Friends" http://devopsenterprise.io/sessions/itil-and-devops-can-be-friends/
"Trust Me: The DevOps Movement fits Perfectly with ITSM" http://www.theitsmreview.com/2014/03/trust-devops-movement-fits-perfectly-itsm/

Implementing The Critical Controls in DevOps (1)

Critical Control	DevOps Practice
CC1: Inventory of Authorized and Unauthorized Devices	Use automated provisioning and configuration management to create an inventory of authorized and unauthorized devices
CC2: Inventory of Authorized and Unauthorized Software	Use automated deployment to maintain inventory of authorized and unauthorized software
CC3: Secure Configurations for Hardware and Software (on Servers)	Use hardened recipes/templates to secure configurations for hardware and software

The CIS Controls for Effective Cyber Defense, aka "the Critical Controls" (https://www.cisecurity.org/critical-controls/) are a set of key recommended practices for implementing a cyber security program. For more on the Critical Controls, how they were developed and how they are used, see: https://www.sans.org/critical-security-controls

Research paper: "Continuous Security: Implementing the Critical Controls in a DevOps Environment" by Alyssa Robinson. Explains how to implement The Center for Internet Security Critical Controls using Continuous Delivery, based on version 5.0 of the Controls

https://www.sans.org/reading-room/whitepapers/critical/continuous-security-implementing-critical-controls-devops-environment-36552

Implementing The Critical Controls in DevOps (2)

Critical Control	DevOps Practice
CC4: Continuous Vulnerability Assessment and Remediation	Continuous testing and CD patching for continuous vulnerability assessment and remediation
CSC6: Application Software Security	Disciplined engineering practices for application software security, including heavy use of automated testing, static analysis and application scanning in the CD pipeline
CSC12: Controlled Use of Administrative Privileges	Infrastructure as code to control use of administrative privileges

This page intentionally left blank.

Automating Compliance Checking: InSpec

- Open source language to write compliance checks that auditors can understand
- Works like Serverspec: test the actual configuration against expected results and report any deviations
- Each test includes severity or risk level, and description information to match against compliance checklist items or regulatory policies
- Testing for Linux, Unix, Windows, AWS, Azure and VMWare

```
control "sshd-11" do
  impact 1.0
  title "Server: Set protocol version to SSHv2"
  desc "Set the SSH protocol version to 2. Don't use
legacy
      insecure SSHv1 connections anymore."
  tag security: "level-1"
  tag "openssh-server"
  ref "Server Security Guide v.1.0" url: "http://..."
  describe sshd_config do
    its('Protocol') { should cmp 2 }
  end
end
```


InSpec (http://inspec.io/) is an open source Ruby DSL for writing declarative compliance checks on infrastructure configuration (InSpec stands for "Infrastructure Specification"). InSpec works similar to the Serverspec acceptance testing framework (it was originally a fork of Serverspec): InSpec automatically checks a server configuration against expected results and reports any variances. Tests can be run on local or remote systems.

Each test defines the severity or risk level of the check. Tests are annotated to match the tests to specific compliance policies or checklist items.

Although sponsored by Chef, InSpec does not need to be used with Chef. Download from GitHub (https://github.com/chef/inspec).

InSpec supports Linux, Unix, Windows, AWS (https://github.com/chef/inspec-aws), Azure (https://github.com/chef/inspec-azure) and VMWare (https://github.com/chef/inspec-vmware) – on cloud platforms, InSpec uses the platform APIs to test services, network objects, storage components and orchestrators.

The authors of InSpec are also involved in Dev-Sec.io ((http://dev-sec.io/)) automated hardening framework project. The hardening framework includes InSpec tests for all of the hardening templates, as well as an InSpec compliance profile for Docker-based on the Docker CIS benchmark (https://github.com/dev-sec/cis-docker-benchmark) and a SSL benchmark test ((https://github.com/dev-sec/ssl-benchmark).

Community project to generate InSpec rules from DoD Security Technical Implementation Guides (STIGs): https://github.com/inspec-stigs/inspec-stigs

Automating Compliance Checking: Chef Compliance

Chef Compliance is a Chef commercial option to automatically verify security and compliance policies

- Chef Compliance Server: centralized scanning across nodes, management of profiles, track statistics
- Pre-built compliance profiles (including CIS benchmarks for Linux, Unix and Windows platforms)
- Based on open source InSpec testing language—you can use this to write your own auditing controls
- Pre-built reports and metrics
- Option to automatically remediate problems found using Chef

Chef Compliance (https://www.chef.io/compliance/) is a commercial offering from Chef which scans infrastructure and reports on compliance issues, security risks, and outdated software. It provides a centrally-managed way to continuously and automatically check and enforce security and compliance policies, in order to achieve what Chef calls "Compliance as Code" and "Compliance at Velocity" (http://pages.chef.io/rs/255-VFB-268/images/compliance-at-velocity2015.pdf).

Compliance profiles are defined in code to validate that systems are configured correctly, using InSpec – an open source testing framework for specifying compliance, security and policy requirements (https://docs.chef.io/inspec_reference.html). You can use InSpec to write high-level, documented tests/assertions to check things such as password complexity rules, database configuration, that packages are installed (or not installed), etc. All of this is based on technology that Chef acquired from a startup called VulcanoSec (http://vulcanosec.com/).

Chef Compliance comes with a set of pre-defined profiles for basic Windows and Linux configuration best practices and common packages (Apache 2, MySQL, PostgreSQL, and SSH), and CIS Level 1 and Level 2 benchmarks for AIX, Apple OSX, CentOS, HP-UX, RHEL, SUSE, Ubuntu and Windows platforms.

Chef Compliance integrates with Chef Server and Chef Automate.

If variances are detected, they are reported to a central dashboard and can be automatically remediated using Chef.

Building Your Own Compliance Reports

- Configuration management tools like Puppet, Chef maintain configuration state and a history of changes
- Use configuration information and change history to create compliance reports
- Tools like **OpenSCAP** can be run to create regular compliance reports
  - Scans operating system and other software against hardening policies based on PCI DSS, STIG and USGCB guidelines
  - Can automatically correct any deficiencies found
  - Can be run from command line and integrated into CD pipeline

Automated configuration management tools like Chef and Puppet maintain a central picture of configuration and a history of changes to configuration. You can use this information, and reporting options provided with these tools, to generate compliance reports.

Automated compliance scanning tools, such as OpenSCAP (https://www.open-scap.org/), can be used to run regular compliance reports and checks. OpenSCAP is a compliance scanner that can be run against Windows, Mac OS X, different Linux distros (Fedora Linux, RHEL 6/7, Debian 8) and other software (Chromium, Mozilla Firefox, Java JRE) to check for a range of different policies including PCI DSS, STIG, USGCB, and others. It can also automatically correct any deficiencies found in the analysis.

Lab #2.5 – Audit with OpenSCAP

Estimated Time: 30 Minutes

Audit the class VM and Bricks container using OpenSCAP

OBJECTIVES

Run the SCAP Workbench to report on the state of our lab VM.

Follow up with a CLI scan and a scan of a container.

PREPARATION

Open a terminal window in the Lab VM

SCAP is the Security Content Automation Protocol developed by NIST (National Institute of Standards and Technology).

SCAP (https://scap.nist.gov/index.html) is designed to be a set of specifications that define a clear, automatable way for security vendors and tools to communicate information about system and application configuration errors and software flaws.

Lab #2.5 Summary – Audit with OpenSCAP

In this lab, you...

- Used SCAP Workbench to audit the VM for compliance with the PCI-DSS v3 standard
- Used the OpenSCAP command line tools to audit the VM and the Bricks application container
- Reviewed the generated HTML audit reports

We've taken a quick tour of various incarnations of the OpenSCAP toolset: from a GUI which can audit a local or remote machine, to tools that we can use to automate scanning both full systems and containers. Seeing how we can run these tools on the command line gives us the key information needed to integrate these with our configuration management system, our CI pipeline, or both.

DevOps and Compliance Summary

- ✓ Compliance rules can be rolled into automated controls in CD pipelines
  - Leverage CD automated control framework for compliance
  - Audit trail from check-in to deployment for traceability
- ✓ Minimize the need for high-cost change control
  - Most changes are small, can be handled as standard pre-approved changes
- ✓ Automate vulnerability management
- ✓ DevOps can be reconciled with ITIL and with prescriptive regulatory frameworks like PCI DSS

Continuous Delivery provides an automated control structure to support compliance requirements. Compliance rules can be burned into stages and checks in CD, providing a complete audit trail of every change from check-in to deployment.

This requires a disciplined engineering approach:

- Every change and fix (to the application or infrastructure) needs to be recorded in a ticket system such as Jira
- All changes need to be made through code
- All code has to be checked in to a source repository, referencing the ticket number
- All changes need to be made through the CD pipeline
- Mandatory compliance checks (reviews, approvals, testing) need to be implemented as part of the workflow and automated where possible

If the engineering and operations teams agree to this approach, you can achieve "Compliance as Code", where you can demonstrate compliance on every change. It may take some explaining to auditors, who are accustomed to waterfall-style/pre-release gates and ITIL-style change management with lots of paperwork. They need to look at ticketing systems, configuration code, and CD audit logs instead.

Change control can be streamlined because most changes are broken down into small, incremental steps which can be handled as "standard" (pre-approved, routine) changes.

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.

Quick Wins – Checklist (1)

Look for changes that you can make in the short term that will have immediate impact

- ✓ Focus on High-Risk Code – know when it is changed, make sure it gets reviewed and tested carefully
- ✓ Secure your Software Supply Chain – know what software components are used, make sure that they are up to date
- ✓ Organize security testing and scanning around engineering Self-Service – make it easy and natural for developers
- ✓ Add automated security testing into the pipeline using a framework like Gauntlt– and get the development team to own these tests

Focus on High-Risk Code

One quick win is making sure that high-risk code changes are consistently reviewed for correctness and defensive coding (data validation, error checking, and logging) – not just for clarity, style and maintainability. Get the team to agree on what code is high-risk and build a workflow to ensure that this code is reviewed before being promoted. Decide on whether code needs to be reviewed – and the results remediated – before it is checked in (as part of a pull request), or before the change is promoted to production.

Automatically scan code check-ins for dangerous functions (crypto, random number generation, etc.) and scan the repositories for hard-coded credentials (passwords and keys).

Review configuration management code (Chef cookbooks, Puppet manifests…), especially for security-related packages/services.

Make sure that reviewers understand how to do code reviews properly, and that they understand defensive coding and secure coding practices – get them training if necessary. High-risk code should be reviewed by (and changed by) senior engineers when possible.

- Consider rotating reviewers or randomly assigning reviewers to prevent the possibility of collusion.
- Consider implementing a rule that high-risk code must be reviewed and approved by at least 2 people.
- Consider using an automated code review tool or Git workflow to manage and track the review and remediation workflow.

Create security-specific checklists for developers and operations engineers to follow when reviewing code. Keep the checklists short. Check for:

- Improper/missing use of encryption
- Plain-text/hard-coded secrets
- Debugging code left in by accident
- Missing data validation
- Missing error handling
- Backdoors (intentional or unintentional)
- Suspect code, including code that is too difficult to follow – whether it is poorly written or intentionally obfuscated

Guidelines on reviewing code for backdoors: http://security.stackexchange.com/questions/3704/how-to-review-code-for-backdoors

Secure your Software Supply Chain

Another good place to start is by understanding and securing build dependencies. This will help to reduce security risks. It will also help you to understand more about the technology, the tools and platforms, the workflows used to build and deploy systems: a valuable first view into the CD pipeline.

It pays to pay attention to the engineering supply chain. This includes:

- Open source application frameworks and libraries
- Open source runtime stacks and databases
- Open source development and CI/CD tools
- Public container images
- Community infrastructure configuration recipes/manifests

Security vulnerabilities inherited from open source code libraries, frameworks, and runtimes create serious security risks for almost all systems today. High-profile vulnerabilities like Heartbleed and ShellShock have shown how dependent almost every organization has become on open source technology, and the system risks that this has created.

Take advantage of tools like OWASP Dependency-Check to identify when a vulnerability is found in an open source component, or when a developer checks-in a component with a known security vulnerability. These checks can be introduced without any major impact on developer workflows, and will help catch/prevent security vulnerabilities early in the cycle, and automatically alert you to new risks in the existing code base, by providing more transparency into the "bill of materials" that make up each system.

If you need to take a harder position on ensuring that developers are not downloading insecure software components, Sonatype offers a Nexus repository health check and firewall which will let you filter out any attempts to download software based on security risk or licensing risk.

http://www.sonatype.org/nexus/2015/11/18/the-nexus-firewall-perimeter-defense-for-software-development/

http://blog.sonatype.com/2012/02/gain-some-insight-with-a-nexus-repository-health-check/

http://www.sonatype.com/assessments/repository-health-check

The same controls should be applied to containers. Make sure to scan container images for known vulnerabilities, or restrict developer access to trusted registries.

Over the longer term, the goal should be to follow Toyota and other Lean leaders in their model to work with "fewer, better suppliers" (although, like Toyota and the auto industry, you also need to watch out for the "single supplier" problem, where problems at a single supplier – like an airbag manufacturer – can cause serious problems).

Identify where teams are using different versions of the same library or framework, and encourage them all to move to the latest "known good" version where possible. Identify where teams are using many different libraries for the same purpose, and work with them to standardize on fewer, well-supported libraries. You need to understand that this won't always be easy – switching out a logging library, for example, might be simple and low risk, but upgrading an application framework can be an expensive and risky move in cases where the team has extended or patched the code on their own, or where the open source maintainers decided to change direction or design in later versions.

Self-Service Scanning and Testing

Providing self-service security tooling to engineers lets them integrate security into their existing workflows (when they need it), minimizing delays and costs, and helps them to take more responsibility for security.

A good place to start is providing self-service static analysis checking (SAST) as done at Twitter and Netflix. Static Analysis checking can be added at different points into engineering workflows: in the IDE, in CI and CD. Make sure to filter out as much noise as possible, and provide clear feedback to engineers. Feed other scanning and test results back to the engineering backlog as bugs to be fixed.

Engineers will accept this kind of checking if it fits naturally into how they work, and if it demonstrates value by finding real problems in their work as they work. Keys to making this work are:

- Return results into engineering tools—in their IDE or directly into their bug tracker/backlog
- Provide feedback quickly—before engineers move on to working on something else
- Take the false positive hit—if results need to be reviewed and confirmed, don't do it as part of the engineering cycle, do it offline
- Make sure that the guidance is clear: Engineers know what needs to be fixed (to the line of code when possible), how to fix it and why

Once engineers take on more of the work of static analysis checking, triage, and remediation, it will free up InfoSec resources to take on other work.

Learn from how Netflix extends the self-service security model through extensive automation and continuous checking.

- Security Monkey and Conformity Monkey check to make sure that new services are configured and deployed correctly
- Scantron automatically identifies new/changed AMIs and runs security testing, will fail if anything serious is found
- Repoman monitors roles, profiles, and users in production and takes away permissions that have not been used

- Monterey automatically identifies new and changed AWS assets and performs vulnerability scanning and testing
- Penguin Shortbread automatically identifies microservices and (re-)evaluates risk based on dependencies

Other work that can be done in a self-service, on-demand way could include building workflows and simple notifications for engineers to request a threat model or secure code review, or a pen test or audit. See the OWASP Appsec Pipeline project for a model and tools to help manage this.

Add security testing into the pipeline

Look for opportunities to inject security testing into development and operations engineering workflows.

Tools like Gauntlt and BDD-Security are designed to make it easier for developers and InfoSec to build a shared understanding of security requirements, as well as making it easier to write tests by abstracting many of the details of working with security tools and dealing with the results.

Start with building a security acceptance smoke test that will be run in the acceptance test cycle – and in production post-deployment – that checks to make sure that the system is setup and running correctly, and that basic features (authentication, access control, and auditing) are working correctly. Include checks to make sure that file and directory permissions are correct, that ports which should be open/closed are open/closed, that SSL and Apache/nginx are configured correctly... This is useful for catching dangerous regressions and configuration mistakes.

Get the engineering teams to own these tests, so that they will ensure that the tests are kept up to date and working, and hopefully build on this starting point.

Quick Wins – Checklist (2)

- ✓ Eliminate Snowflakes: use Infrastructure as Code to standardize and audit configurations and enforce hardening policies
- ✓ Secure the CD pipeline from start to finish (from check-in to deployment)
- ✓ Use Containers to isolate workloads—but do it carefully
- ✓ Use RASP or other Run-Time Defense technology—but do it carefully

Infrastructure as Code: Eliminate Snowflakes

Infrastructure as Code using tools like Puppet, Chef or Ansible ensures consistency across managed systems: you can confidence that systems are set up the same way. And this allows you to enforce policies consistently across systems. These tools:

- Simplify, standardize and speed up provisioning of systems, and make this transparent
- Continuously and automatically scan and monitor the configuration
- Automatically detect (and correct) unauthorized changes to configuration, and prevent configuration drift
- Automatically detect (and minimize) differences between development, test and production environments
- Audit configuration changes and patching, so that you know where you have vulnerabilities and can confirm when they have been patched

Because configurations are defined – and changed – through code, you can introduce security early through code reviews, before configurations are pushed out to production. Build a secure workflow to ensure that changes to configuration are reviewed before being promoted to production.

Secure your CD Pipeline

In order to rely on the integrity of your CD pipeline and the chain of custody, you need to secure the CD pipeline from start to finish – from check-in of changes through to deployment. This is important for security, compliance and the overall integrity of the process. Treat these tools and the infrastructure that they run on as part of your production infrastructure. They need to be secure, reliable and continuously monitored.

Start by Threat Modeling the Configuration Management system (Chef, Puppet, Ansible…), and your Continuous Delivery pipelines and toolchain. Look for weaknesses and gaps.

Make sure that you understand – and control – what is on premise and what is in the cloud. It is easy for development teams to set up cloud-based repos and CI/CD services which could be extending the attack surface of your systems into the public cloud, or accidentally exposing Intellectual Property or secrets.

Some steps to take:

- Do not allow anonymous, unauthenticated check-ins or anonymous push of changes
- Trace handling credentials and secrets – ensure that they are securely stored and used
- Review and harden the configuration of the toolchain – the source repo and binary artifact repos, and especially the CI/CD server and deployment tools (most of them are not secure by default)
- Scan and harden the runtime environment for the entire pipeline
- Periodically check the audit trails to ensure that they are complete and consistent

Use Containers to Isolate Workloads

Developers and operations will want to take advantage of containers (like Docker) to streamline system provisioning, packaging, and deployment, in development, in test and Continuous Integration/Continuous Delivery, and in production.

Containers, if used properly, can provide some quick security benefits:

- Standardized configurations
- Simplified, automated deployment
- Reduced attack surface

It's important to enforce some basic security practices and controls when working with containers:

Make sure to establish trust with images:

- Use official repos and trusted registries
- Scan images for vulnerabilities and out-of-date software
- Build a review workflow into the push stage

Take advantage of automated tools like Docker Bench to scan containers and check configurations for least privilege and best practices.

Consider using VMs to provide additional isolation, especially in multi-tenant applications.

RASP and Runtime Defense

Runtime Application Self-Protection (RASP) and other runtime defense solutions may offer a way to reduce security risks for third party and legacy apps (mostly Java and .NET), as well as serving as a defense-in-depth control for rapidly-changing systems in Continuous Delivery/Deployment:

1. Running in monitoring mode, they can provide visibility into attacks and runtime errors
2. This attack information can be analyzed across applications to find trends and for forensic purposes
3. Running in protection mode, RASP solutions can block many common attacks and known vulnerabilities in runtimes and in popular libraries, with low false positive rates
4. SAST and DAST results can be used to create "virtual patches" for many RASP solutions in the same way that patches can be made to WAFs on the fly

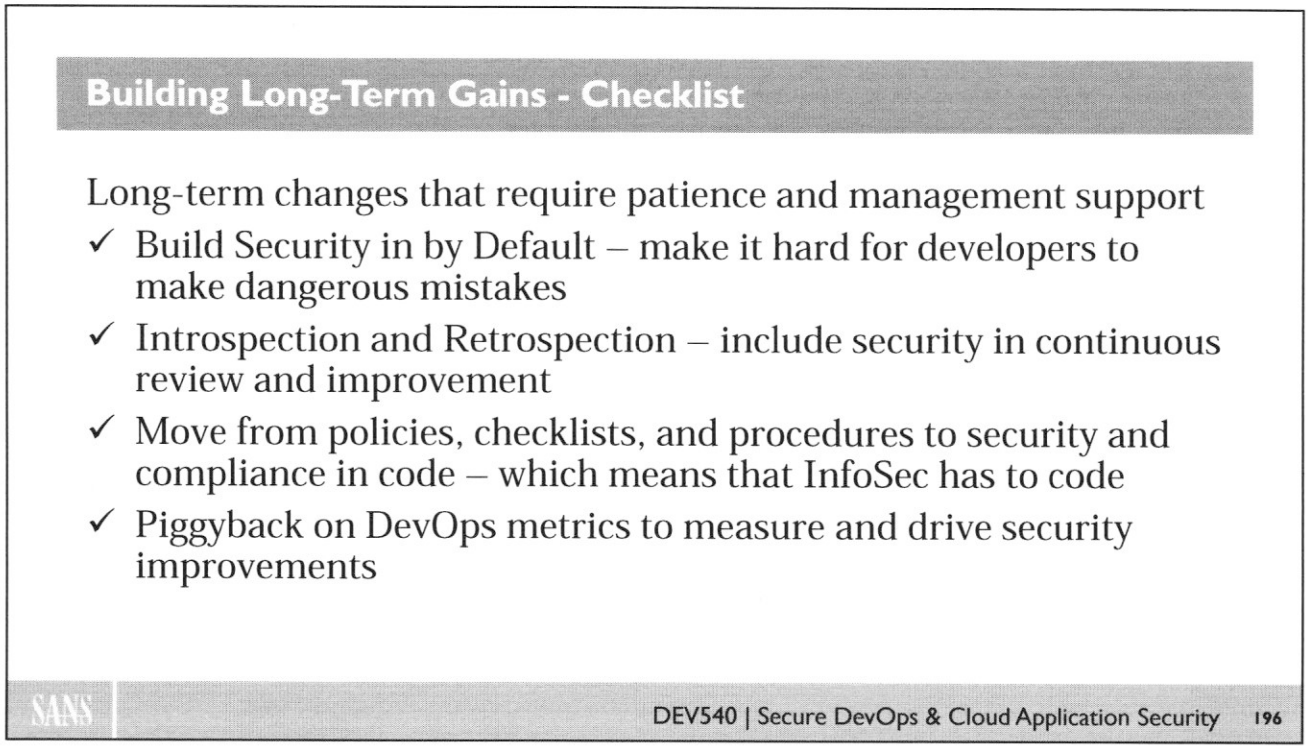
RASP can be implemented quickly and can scale across multiple applications and services (especially in a microservice environment).

This is an emerging technology that should be closely watched. Although some analysts from Gartner are firmly behind the approach, not everyone is convinced that RASP is reliable or safe:

See "RASP is Dead in the Water," by Richard Quinn http://richard-quinn.com/2014/08/22/appsec-rasp-is-dead-in-the-water/

"Gartner analysts debate value of perimeter firewalls vs. 'Runtime Application Self-Protection'", June 2014:

http://www.networkworld.com/article/2365739/security0/will-perimeter-firewalls-give-way-to-rasp.html



Build Security in By Default

Building security into engineering, making engineering safe by default, as we've seen from the case studies in this course, is fundamental to the security and engineering programs at organizations like Google, Amazon, Facebook, Etsy, PayPal, and Netflix.

This will require a significant change to how engineering is done in many organizations, and real commitment and buy-in from engineering and management to accomplish.

This is a long game to play. It will require training and time and funding and patience.

Introspection and Retrospection

Understand and leverage the continuous review and improvement practices and feedback cycles in Agile and DevOps. Make sure that security concerns are included. Use data breaches, security incidents and audit failures or pen test failures as a chance for the team to learn and improve through Blameless Postmortem analysis. Why did these problems escape the team's reviews and testing and other checks? How can we improve our preventative and detective controls?

Keep asking questions and keep looking for answers until you get to the Root Cause(s) – there is usually more than one Root Cause. Then work with the team to address weaknesses in the organization, in training, in design, in management.

Try running your security program the same way that DevOps teams work: iteratively, running experiments, measure the effects, and adjusting, adapting in response to feedback. Improvise and improve. Learn how to pivot and change focus or direction.

Move from Policies to Code

One of the fundamental changes in DevOps is that almost everything is code:

- Infrastructure configuration is done in code
- Testing, including security testing, is done through code
- Packaging and deployment is done through code
- Vulnerabilities aren't compliance violations tracked in a spreadsheet—they are bugs in code that need to be fixed
- Auditing is done on code through code

This means that security needs to change from a compliance/.governance role focused on enforcing policies and following checklists, and relying on control gates like manual security audits and pen tests, to an engineering role that is centered on understanding, reviewing and writing good, safe code:

- Making sure that code is managed properly through disciplined version control and Continuous Integration and Continuous Delivery
- Making code (application code and infrastructure code) secure and safe by default using safe libraries, templates, frameworks, and headers, so that is easier for engineers to do the right thing from the start
- Identifying high-risk code so that changes to this code can be closely monitored
- Writing automated tests to test security controls and to make sure that the system is set up correctly and safely
- Designing and building self-service security tools for engineers
- Helping to build up the CI/CD pipeline, automating and standardizing and streamlining build, test and deployment steps
- Reviewing code to make sure that all of this is done properly
- Not just finding security bugs in code, but fixing them or helping engineers fix them

Policies should be described and enforced through automated asserts and tests where possible. "Do not do xxx" in a legal document becomes "Does not do x" in an automated test. This will take time and training and reinforcement to get the necessary buy-in from designers and management and change their thinking and priorities—or a crisis, like what happened at Twitter.

And this means that security analysts and testers need to be able to code. At Intuit, for example, security engineers are encouraged to start coding—at least something simple—in their first week. If they don't know how to code, they are given online training through Code School (https://www.codeschool.com)

Metrics, Metrics, Metrics

DevOps teams are obsessed with metrics. Take advantage of this.

Decide what security metrics matter to you, to management, and to the DevOps teams. Some candidate metrics:

- Number of serious vulnerabilities – open and fixed. Open vulnerabilities are operational risks. Fixed vulnerabilities tell a story about managing this risk. What percentage of vulnerabilities are fixed? Is this getting better —or worse—over time?
- How long serious vulnerabilities have been open – The window of exposure to attack. Is this window getting shorter? If not, why not? What do we have to do to close it?
- Change Lead Time – The time from when a change is made to when it is deployed to production. This is a key metric that DevOps teams optimize for (minimizing this time is one of the key goals of CD), and it can make an important difference to minimizing the Mean Time to Repair a vulnerability.

- Mean Time to Repair – The full Cycle Time from discovering a vulnerability to creating a patch and getting it successfully deployed.
- Test coverage – What level of automated test coverage is acceptable to everyone for high-risk code?
- SAST and DAST findings – What's the bar that the code must pass in scanning, what findings will fail the build vs what findings will be accepted, at least for the time being?
- Pen test results and audit findings – Highlighting weaknesses in controls and processes. If you are doing everything right, then problems should have been caught and corrected before a pen test. This means that if there are any serious findings, you aren't doing everything right. Use this as an opportunity for the team to reflect and improve, leveraging Blameless Postmortems.
- Run-time production attack metrics: what attacks are most active, what weaknesses are bad guys looking for? Are you protected from these attacks?

Measure – and use these metrics to identify areas where you need to improve, and where you are improving.

Course Roadmap

- Section 1: Introduction to Secure DevOps
- *Section 2: Moving to Production*
- Section 3: Moving to the Cloud
- Section 4: Cloud Application Security (Part 1)
- Section 5: Cloud Application Security (Part 2)

SECTION 2

- Secure Infrastructure as Code
  - Puppet Drill Down
  - Lab #2.1: Manage Configuration with Puppet
- Securing CM and CD Pipelines
- Containers and Container Security
  - Securing Docker
  - Lab #2.2: Audit Docker's Security
- Monitoring and Metrics
  - Lab #2.3: Monitor with Dashboards
- Managing Secrets in CD
  - Lab #2.4: Protecting Secrets with Vault
- Compliance as Code
  - Lab #2.5: Audit with OpenSCAP
- Going Forward
  - Quick Wins and Long-Term Gains

This page intentionally left blank.