Pragmatic Cloud Security: Hands-On Turbocharged Edition

Rich Mogull
@rmogull
rmogull@securosis.com

What to Expect

- Four days of training…
  - in four hours
- Nearly all labs
- Some of you won't be able to keep up
  - That's okay, you have the materials to go through later
  - Doesn't mean you're dumb, just that it's outside your experience
- We probably won't get through everything. This really is material taken from 4 days of training, slimmed down as best we can

Securosis

Reminder: We are not responsible for your AWS bill if you forget to delete things after class finishes.

Really… costs are minimal for our 4 hours, but will add up if you leave things running.

Labs

- All labs are how it's done for real, these aren't merely academic exercises.
- Part 1: we will configure a production-quality account with multiple virtual networks and core security controls.
- Part 2: we will build a deployment pipeline, integrate it into our existing application stack, and code a variety of security automation controls.
  - Most programming labs and examples are in Ruby, with some use of Python.
  - We provide enough code that those of you with less experience can still participate.
- Nearly all labs are in Amazon Web Services
  - AWS is 85% of the market and the most mature (for now)
  - We will reference Azure and Google Compute Cloud, but all the labs are in AWS due to logistics as much as anything

There is minimal hand holding.
We don't have slides for every step.

Securosis

Architecting with Accounts

Securosis

Root Account Security

Securosis

Important Lab Note

All labs should be performed in the us-west-2 (Oregon) region unless otherwise specified

Securosis

AWS Root Account Security Lab

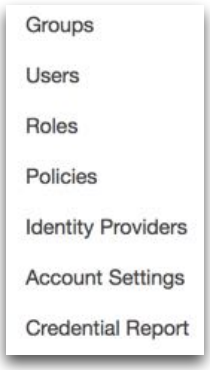
- Enable MFA
  - Authy is free, easy, and on iOS and Android
- Disable Access Keys
- Create new IAM user with the “Administrator” managed policy attached
- Use that user for the remainder of class

Securosis

Identity and Access Management

Securosis

AWS IAM Primitives

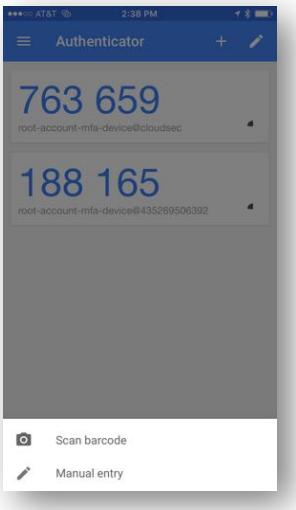


- Users
  - An individual/account capable of logging into AWS
- Groups
  - A collection of users
- Roles
  - Temporary permissions other entities can assume
  - Core to internal AWS services and federation
- Policies
  - The core of IAM
  - Managed and Inline options
- Identity Providers
  - SAML support
- Console URL
  - How IAM users can log into your account

Securosis

Credentials

- Access/Secret Keys
  - For API calls
  - Uses Signature 4 HTTP request signing
- Username/Password
- ***MFA***
- Host keys (Linux)
- Administrator Password (Windows)
- SSH keys (CodeCommit today, maybe more later)



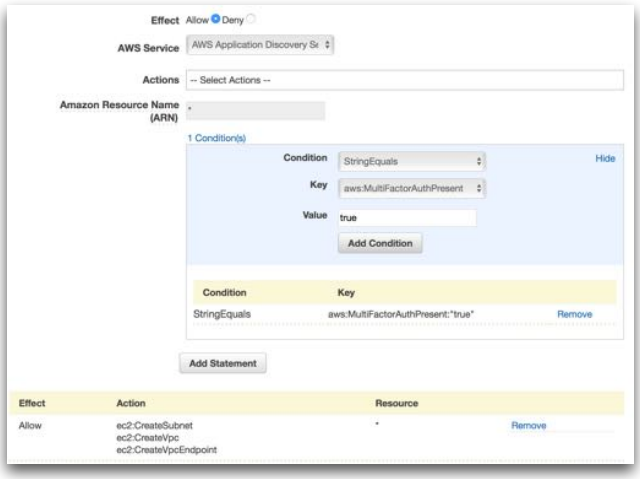
Securosis

Policies

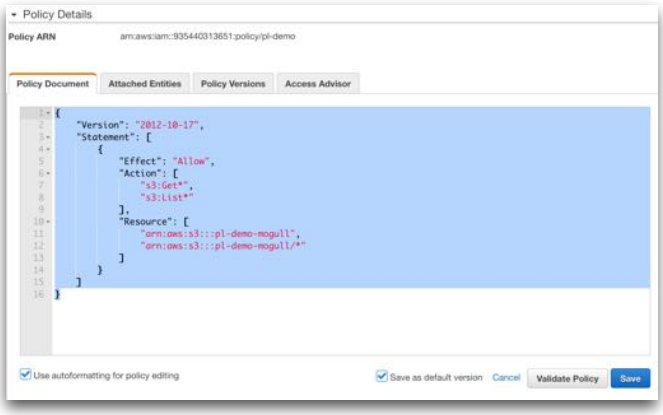
- JSON, but a specific policy language
- Inline apply to a single entity
  - Generally limit use
- Managed policy updates roll out to all linked entities
  - Supports versioning
- Advanced capabilities for dynamic use
  - Especially conditionals

Securosis

Creating Policies



Policy Generator



Hand-Editing

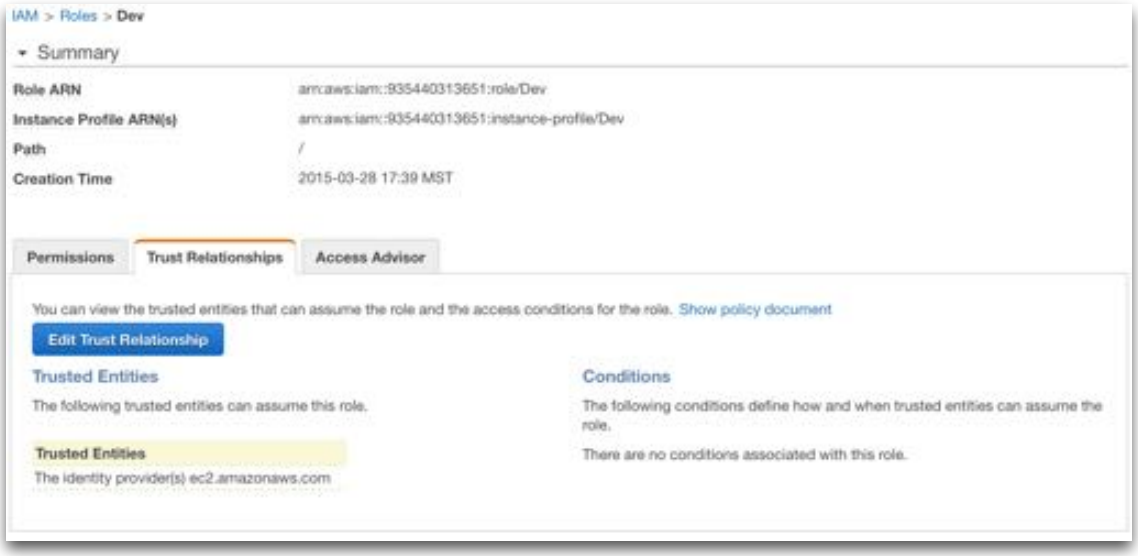
Securosis

Testing Policies

https://policysim.aws.amazon.com

Securosis

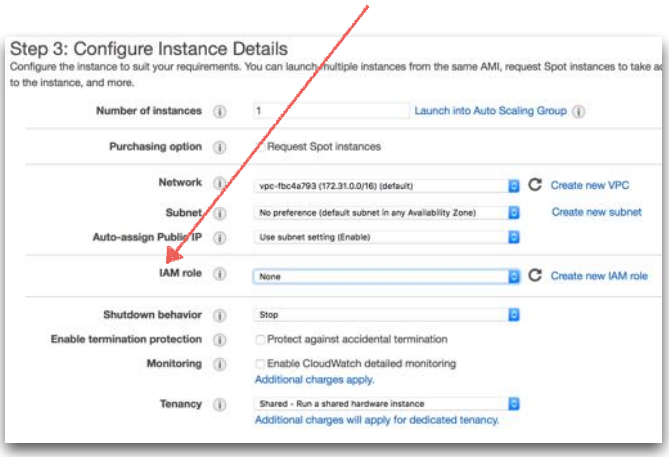
Using Roles (Demo)



Securosis

Lab: Create an Instance Role

- Create a new role for EC2 Instances
- Name it “Dev”
  - We need this later, do NOT use a different name
- Assign the *PowerUserAccess* role
- Launch an instance with the role assigned
  - Log in
  - From the command line, type:
    - `aws ec2 describe-instances`
    - `aws iam list-users`
  - It work? Terminate



Securosis

Using Conditionals (Demo)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1424727721000",
            "Effect": "Deny",
            "Action": [
                "ec2:*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/SecurityStatus": "IR"
                }
            },
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Securosis

Lab: IAM Policies with Conditionals

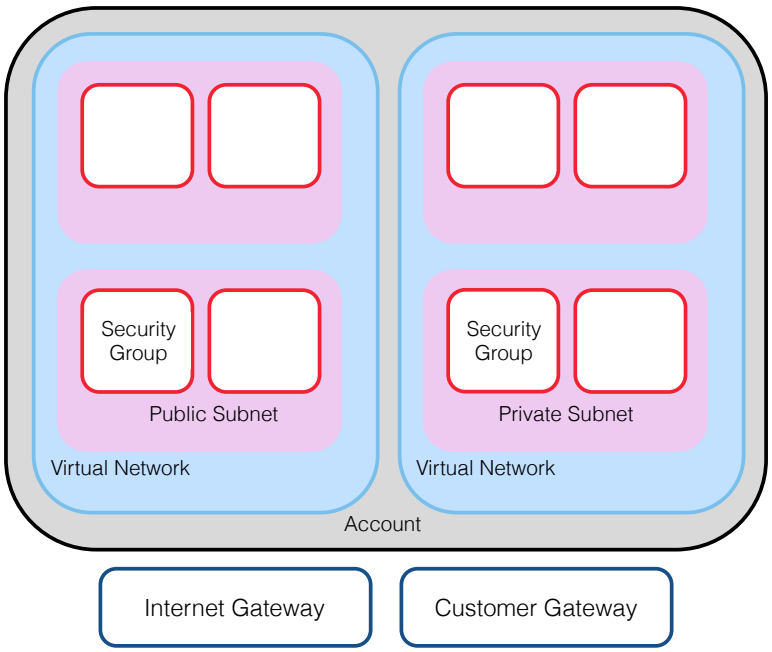
- Write an IAM policy that restricts a user to only a single region
- Write an IAM policy that allows an administrator to manage their own MFA, and requires MFA to log into the console.

Securosis

Cloud Network Security

Securosis

AWS Network Primitives



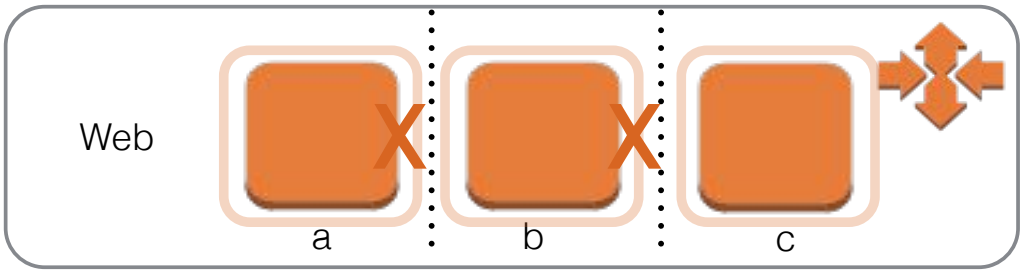
- Elastic Network Interface
- Route Table
- ACLs
- Service Endpoints

Internet Gateway

Customer Gateway

Securosis

Network segregation by default



Type	Protocol	Port Range	Source
SSH	TCP	22	68.2.174.98/32
HTTP	TCP	80	0.0.0.0/0

Granularity of a host firewall with the manageability of a network firewall

Securosis

How Amazon's Network Works

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

Securosis

Fix the VPC

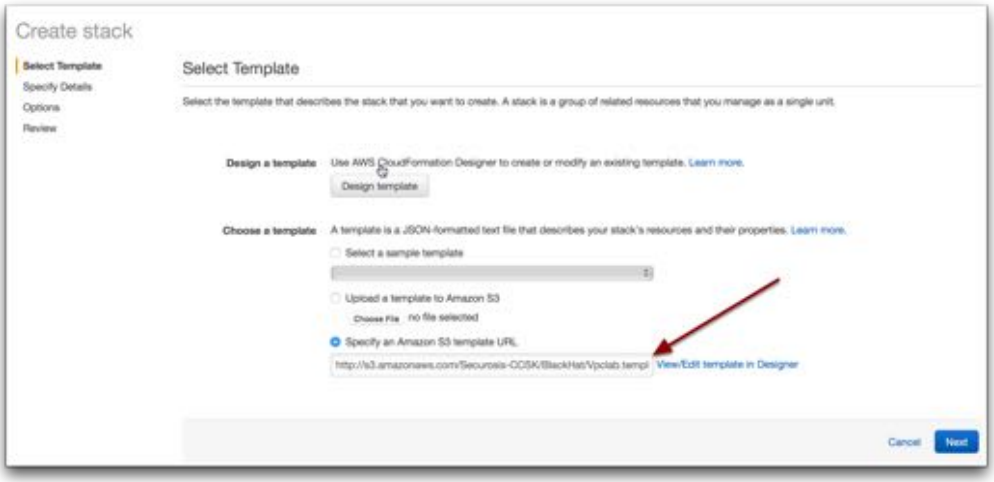
Securosis

Fix the VPC

- Go into CloudFormation
- Launch the stack using the template:
  - http://s3.amazonaws.com/Securosis-CCSK/BlackHat/Vpclab.template
- SSH to the public instance
- SSH to the private instance

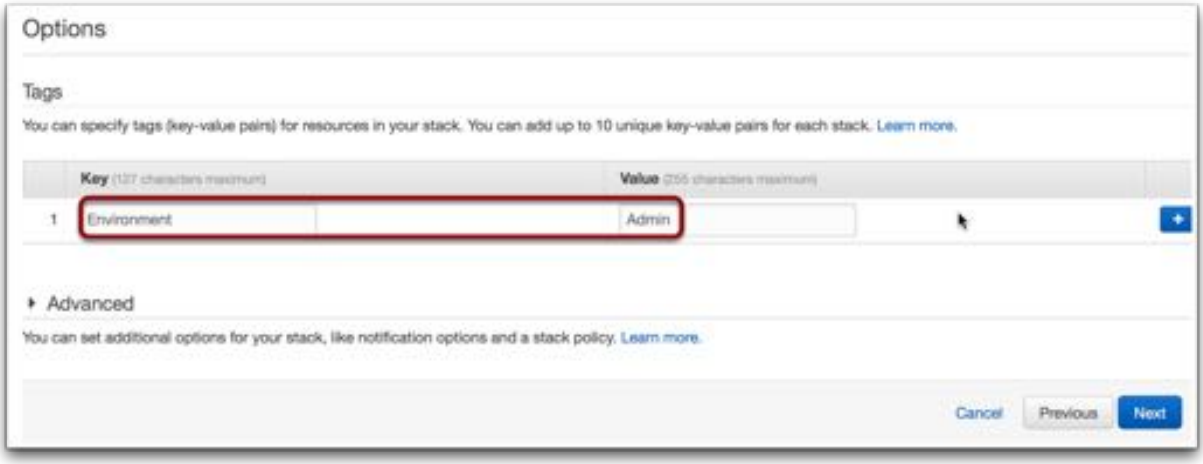

```
{
  "AWSTemplateFormatVersion": "2010-09-09",

  "Parameters" : {
    "KeyName" : {
      "Description" : "The EC2 Key Pair to allow SSH access to the instance",
      "Type" : "String"
    }
  },
"Resources": {
  "LabVPC": {
    "Type": "AWS::EC2::VPC",
    "Properties": {
      "CidrBlock": "10.0.0.0/16",
      "InstanceTenancy": "default",
      "EnableDnsSupport": "true",
      "EnableDnsHostnames": "true",
      "Tags": [
        {
          "Key": "Name",
          "Value": "CloudSec Lab"
        },
        {
          "Key": "Group",
          "Value": "Vpclab"
        }
      ]
    }
  },
  "subnet667a7420": {
    "Type": "AWS::EC2::Subnet",
    "Properties": {
      "CidrBlock": "10.0.0.0/24",
      "AvailabilityZone": "us-west-2c",
      "VpcId": {
        "Ref": "LabVPC"
      },
      "Tags": [
        {
```


Securosis

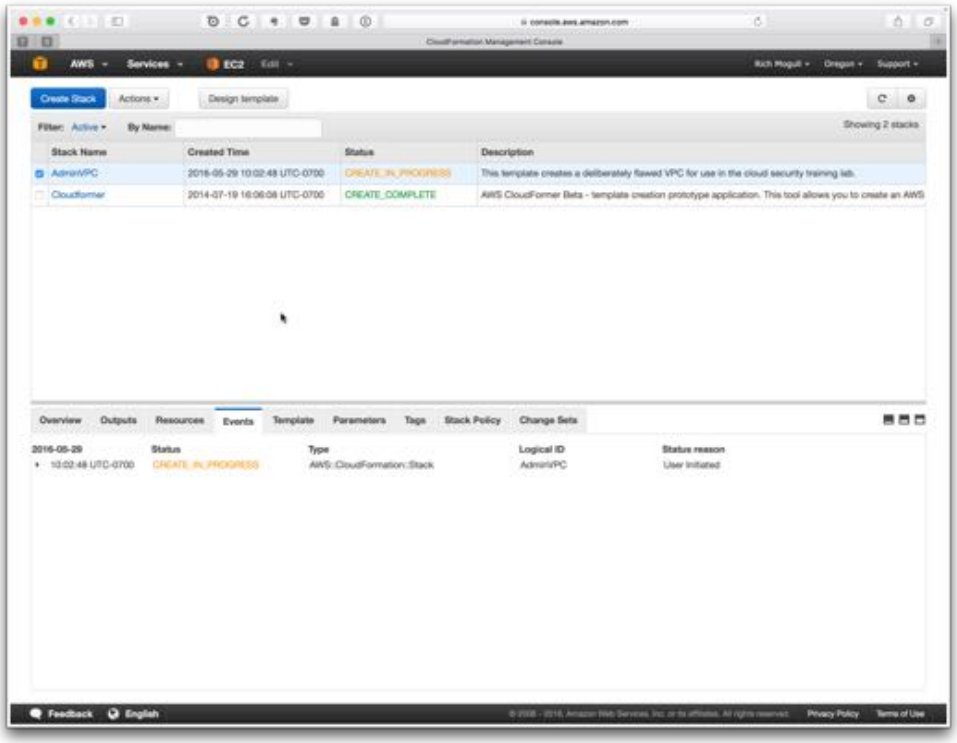


Securosis

Securosis



Securosis



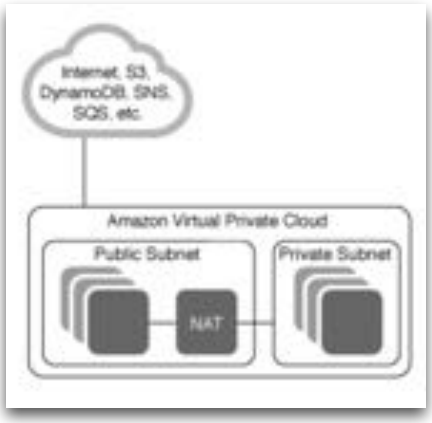
Securosis

VPC Peering

Securosis

VPC Peering Lab

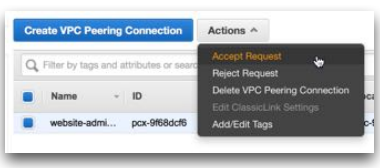
- Allocate a new Elastic IP
- Create a second VPC with public and private subnets
  - Use the VPC wizard
  - Name it “website”
  - Use a CIDR of 192.168.0.0/16
  - Create /24 subnets
- Launch a default Amazon Linux in the *private* subnet
  - With security group called “temp” allowing port 22 from your IP



Securosis

Peer the VPCs

- Name the peering connection “website-admin peer”
  - Doesn’t matter which VPC is in which place
- Accept the request
  - No IAM needed since this is all in your account
- Adjust your routing tables on each side to support the peering connection
  - You could use ACLs to get fancy, but that will also probably break things



Securosis

Connect to your instance in the *Private "website" subnet* from your *Admin jump box*

Pro tip- you need to add the Security Group ID to the website security group for the admin group using cut and paste.

Securosis

Now terminate the instance. Hopefully you weren't emotionally attached.

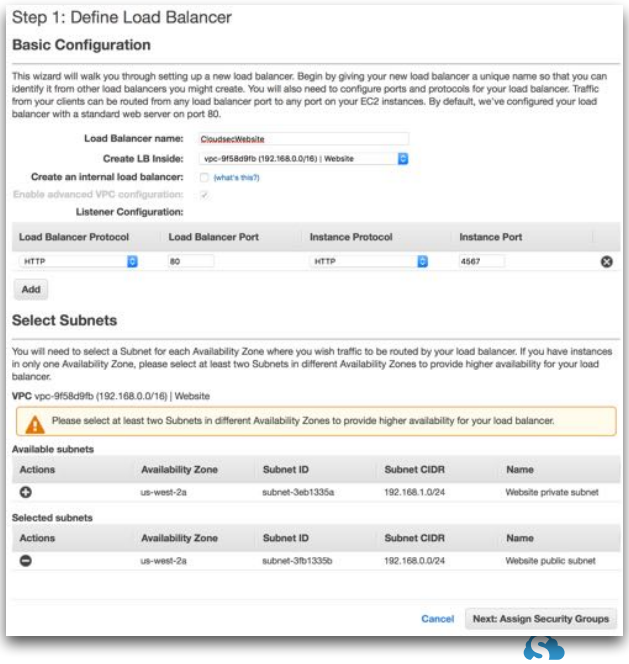
Securosis

Elastic Load Balancers and Auto Scale Groups

Securosis

Lab: Auto Scale Group behind an ELB

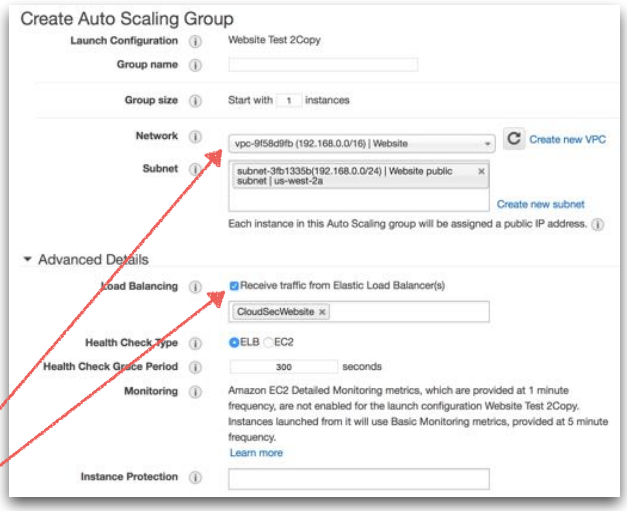
- Create a Load Balancer
  - Name: CloudSecWebsite
  - Pick the *website* VPC and the public subnet
  - Remap port 80 to port 4567
  - Select the *public* subnet
  - Create a new security group
    - Name: Website ELB
    - Port 80 wide open
  - Health Check: TCP 4567



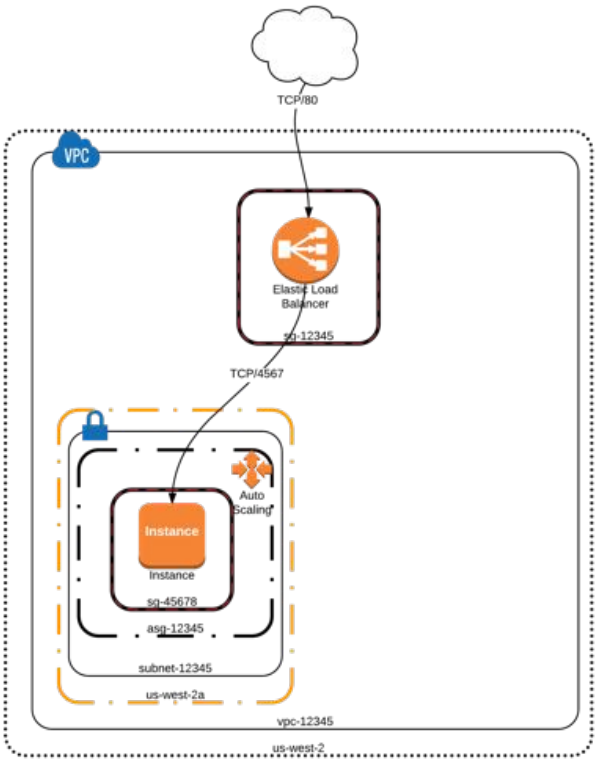
Securosis

Lab: Auto Scale Group behind an ELB

- Create a Launch Configuration
  - Use ami-b73efed7
  - Assign the “Dev” role
  - Assign a public IP address to every instance (under the advanced tab)
  - Security Group
    - Name “Website Server”
    - Port 4567 open from sg-[Website ELB]
- Create an auto scale group with min/max/desired of 1
- Name it “Website”
- Apply a tag “Environment” with value “Production” and apply to new instances
- Choose your Website VPC and public subnet
- Receive traffic from your ELB (Advanced)



Securosis



Securosis

Go to your ELB URL

Securosis Advanced Cloud Security Training Server

I may be ugly, but I get the job done

Current Instance ID: i-0f41d67d231c939d9

Current Image ID: ami-c229c0a2

Securosis

Service Endpoints

Securosis

Service Endpoints

- Directly route traffic from a private AWS network to an Amazon service.
- Right now, only have S3, but more coming
  - Without them, instances need public Internet access
- Also reduce costs since you aren't routing to the Internet
  - Managed by VPC networking and the mapping service, which handles translation of the URL to connect to the AWS service
- Most services require Internet access (e.g. Dynamo, SQS)

Securosis

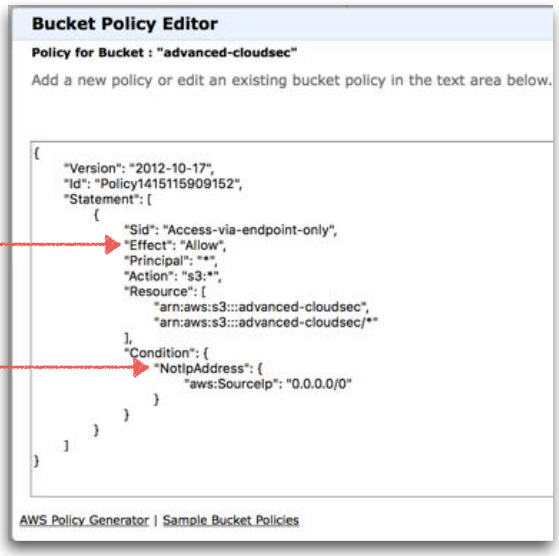
Service Endpoint Lab

- Create an S3 Service Endpoint
  - In your *Website* VPC
  - Go back to your web site ELB
  - See the boom?

Securosis

What's Going On

- The service endpoint routes traffic internally to S3
  - Only works in the same region
  - Our app pulls a URL from a configuration file in S3
- S3 sees it as internal, not as normal IP access
  - Our bucket policy on the config file only allows non-IP-based access
  - If it's inside your own account, you can use an IAM role instead



Securosis

Monitoring and Logging

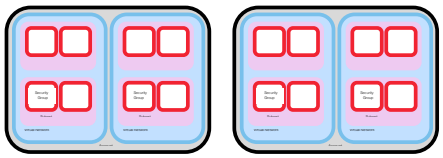
Securosis

What's different

Management Plane

Velocity

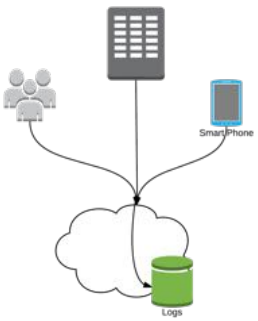
Distribution/Segregation



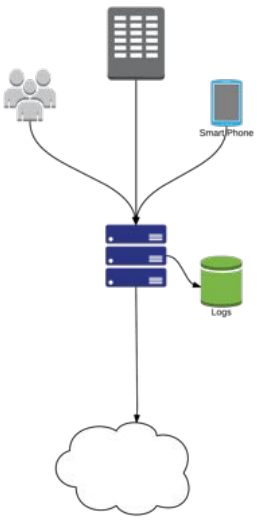
Securosis

Management Plane Security Monitoring

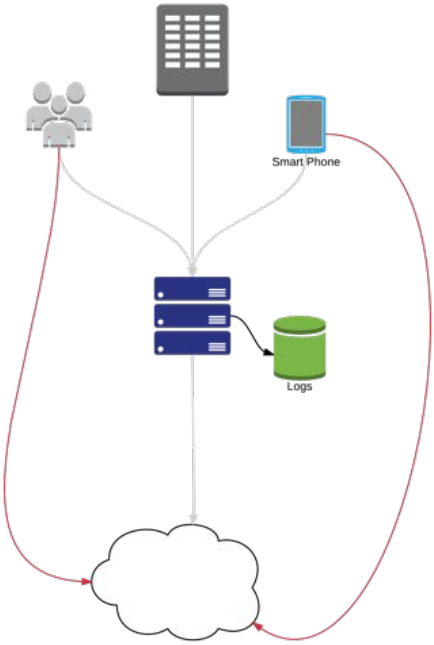
Direct



Proxy

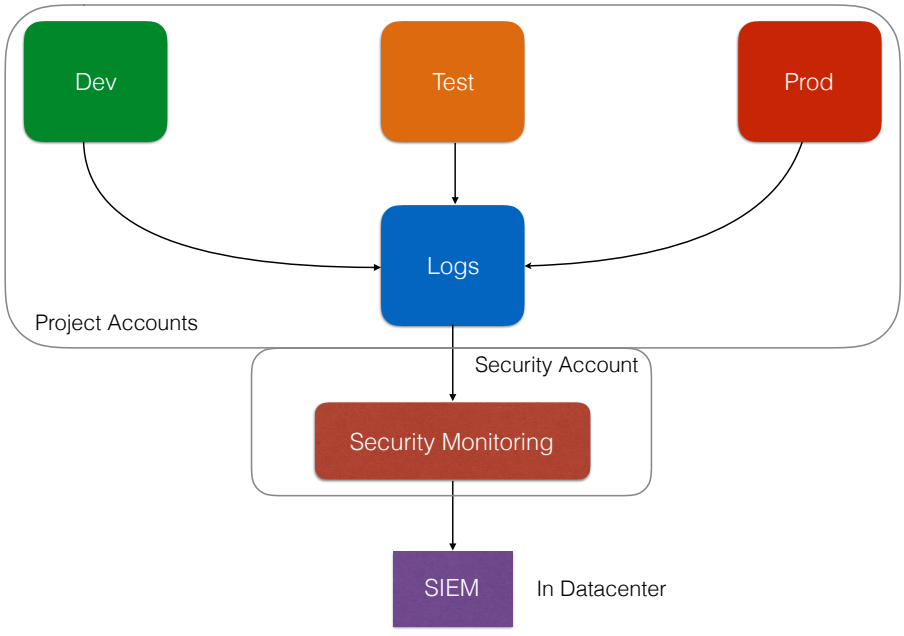


Securosis



Securosis

Cascading Architecture



Securosis

Important Points

- Filter logs down before you transfer over the Internet. All cloud providers charge by the byte, and your goal is to manage costs instead of just pushing everything into the SIEM
  - Management plane logs will have less data than instances and other sources, but it can still add up.
  - We'll cover filtering priorities in a bit.
- You need to correlate what accounts you have to what is being logged. This varies by provider, but typically you compare billing and project IDs to those in your logs.
- Project logs should be readable by project admins only within their project storage.

Securosis

AWS Core Tools

CloudTrail

- *Nearly* all API calls
- Stored in S3
- Can push to CloudWatch
- S3 logs written every 5 minutes
- Can have up to a 15 minute delay
- JSON format
- Integrates with SNS to notify when logs written
- 7 days visible in the console

CloudWatch

- AWS's native log service
- First job is monitoring for performance, auto scaling, etc.
- Can accept *Log Streams* from other sources, like CloudTrail
- Visible in console, stored internally, export via API or to S3
- Supports *Events and Rules*
- Native filtering for alarms to SNS
- Can accept syslog or other arbitrary streams

Config

- Partial service coverage with emphasis on EC2 and IAM.
- Tracks configuration state over time and what assets are affected by what changes.
- Stored in S3
- Visualize in console
- Can enforce *Config Rules*

Securosis

Lab: Initial Configuration

- Enable CloudTrail in all regions
  - Send to a secure S3 bucket with restrictive IAM rights
  - Don't worry about encrypting, but it's an option for highly-sensitive environments.
- Stream to CloudWatch
  - You'll see why later
- Enable Config
- Turn on VPC flow logs (in the VPC section of the console)

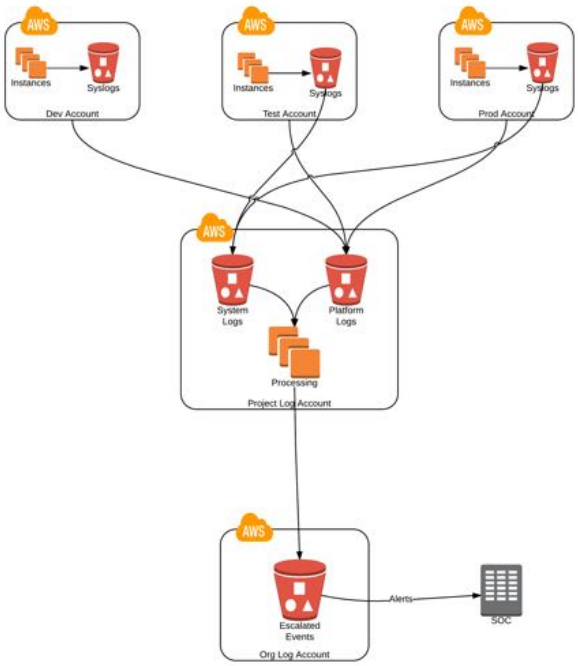
Securosis

Server and Application Logs

- Can be a tougher problem, and depends a ton on your specific requirements
  - Instances/virtual machines have shorter lifespans, and different ones may share the same IP address and name throughout the course of the day.
  - Becomes critical to capture the actual instance identifier
- Quite likely a higher volume of data, so cost management is a bigger issue
- Key issue is build your own or leverage your platform

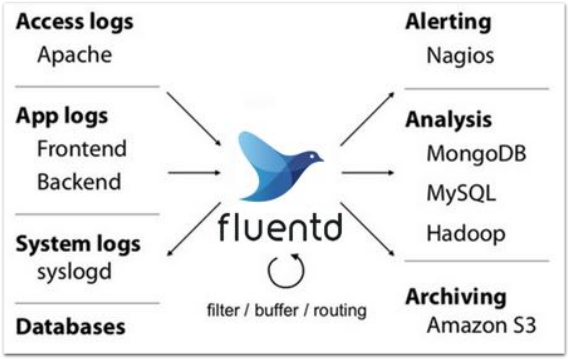
Securosis

Cascading System Logs



Securosis

Lab: Cascading System Logging



- We will set up a central log server using fluentd
- We will modify rsyslog to add the instance ID to log messages and send them to the central logger
- We will configure fluentd to filter for security events and save those to S3
- This creates a skeleton log framework that can scale to far larger and more complex scenarios

Securosis

Set up your log server

- Log into your *Admin* instance
- Install fluentd
  - `sudo curl -L https://toolbelt.treasuredata.com/sh/install-redhat-td-agent2.sh | sh`
- Alter the configuration file to save events to S3
  - See the Cheat Sheet for the specific commands and template
- Open port 5140 from 0.0.0.0/0
- Start it

Set up an instance to push logs

- Launch a new instance, or use any other running instance
- Log in and…
  - Pull the instance ID using the metadata service
    - Yes, you can get it from the console, but this is how you would need to get it to script things
  - Edit rsyslog.conf to push external logs and insert the instance ID into the message
    - You could also modify the hostname instead
    - There are a lot of options for this part of the problem
- sudo service rsyslog restart

Generate Some Logs

- A regular instance won't create enough log files to trigger the push to S3
- From the cheat sheet, create a log generator
  - Create an empty file in your home directory named "log_generator.rb"
  - Paste in the content from the cheat sheet
  - `ruby log_generator.rb`
- Run the log generator and keep an eye on your S3 bucket

Securosis

Now Filter the Messages

- Fluentd supports all sorts of filtering, parsing, and other log flow
- We can use this to store local project logs, while also forwarding on security-sensitive logs
- Un-comment the simple filter in our configuration file
  - This will now only push messages from our filter generator into S3
  - Feel free to play with something more complex

Securosis

Filtering and Alerting

CIS Benchmarks for AWS

Convert for your provider of choice

- Unauthorized API calls
- Management console log in without MFA
- Any IAM policy change
- CloudTrail configuration changes
- Management console authentication failures
- Disabling or scheduling the deletion of a KMS Customer Managed Key (encryption key)

- Any use of the root account
- Any S3 bucket policy changes
- AWS Config configuration changes
- Security group changes
- Network Access Control List changes
- Network gateway changes
- VPC changes (e.g. subnets, routing tables, service endpoints)

Filter to focus security for cascading logs, alert for security response

Securosis

AWS Options

- CloudTrail -> CloudWatch
  - Metrics Filters
  - Events (Rules)
    - By far currently the fastest option
    - Can trigger Lambda functions
- Config
  - Rules

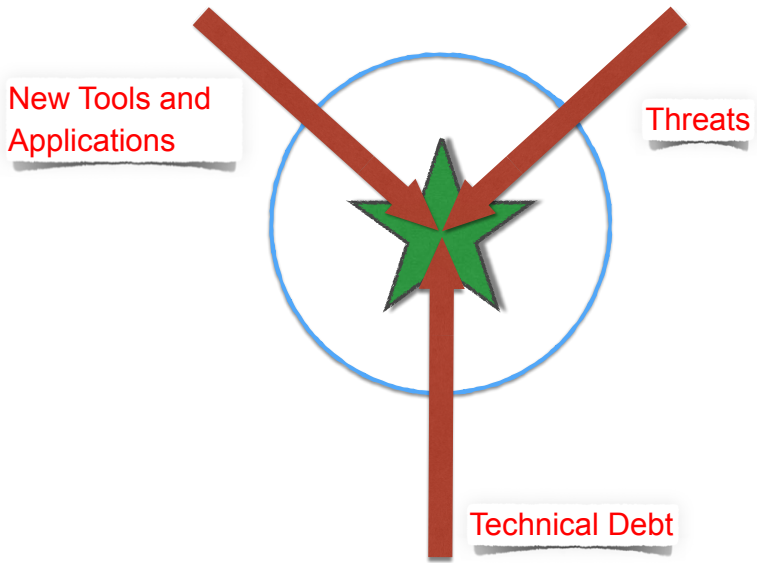
amazon web services™

Securosis

Introduction to DevOps

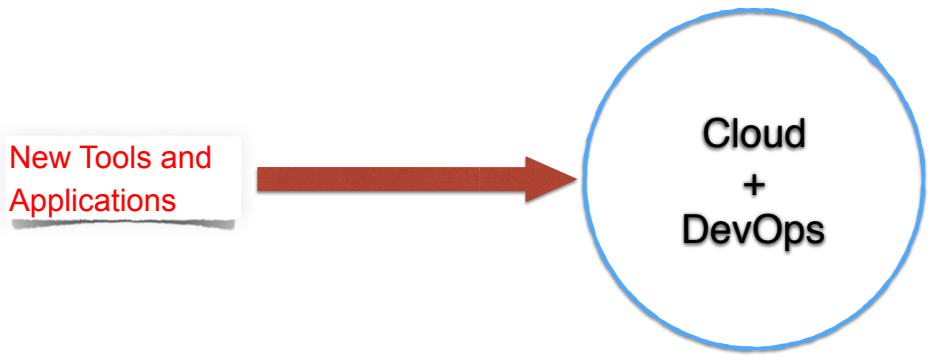
Securosis

The Problem



Securosis

The Problem



Securosis

Cloud and DevOps

- Cloud is a new operational model.
- It requires a re-thinking of fundamental architectures.
- DevOps is a new operational framework, highly attuned to cloud.
- Both shatter existing security approaches.

Securosis

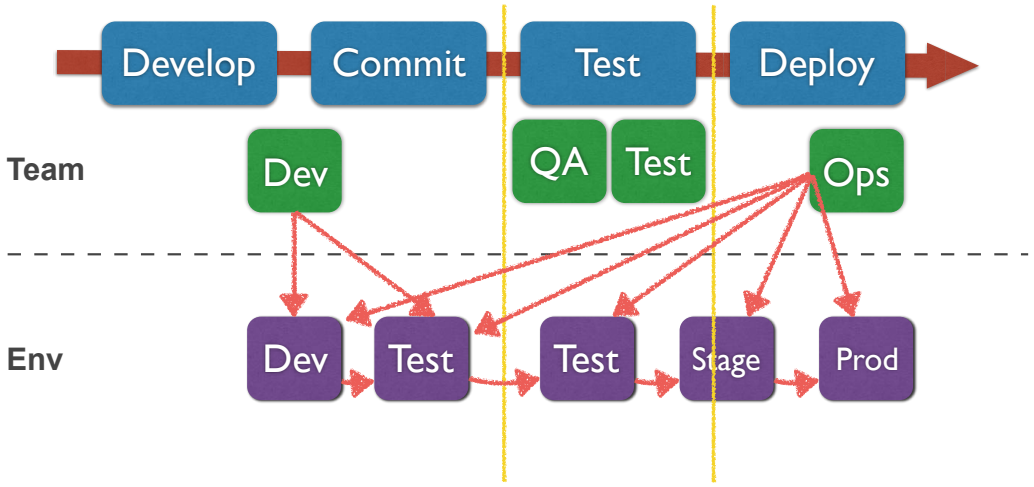
Why DevOps?

Securosis

Application Deployment

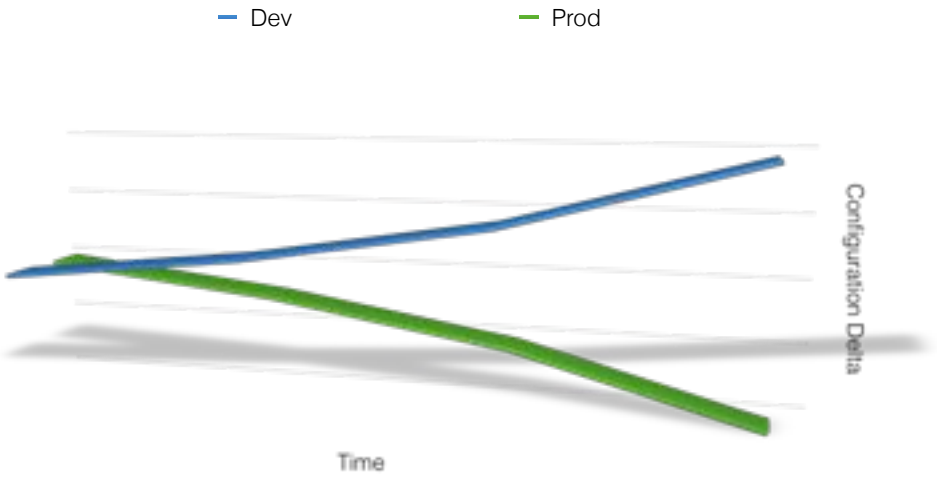
Securosis

Complexity Breeds Error



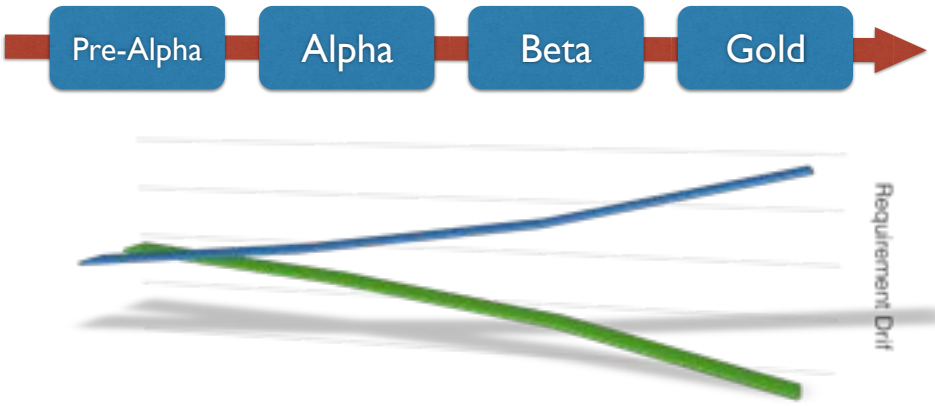
Securosis

Environments, Requirements, and Configurations Drift



Securosis

Cycle Times Matter



Agile? Waterfall?

All the same…

Securosis

The Dev and Ops Problem

- Configurations drift over time.
- The less au
- Manual inte on.
- Environmer
- Different teams work in different environments.
- The longer the update cycle, the greater the chances for error.

No standards

=

More complexity

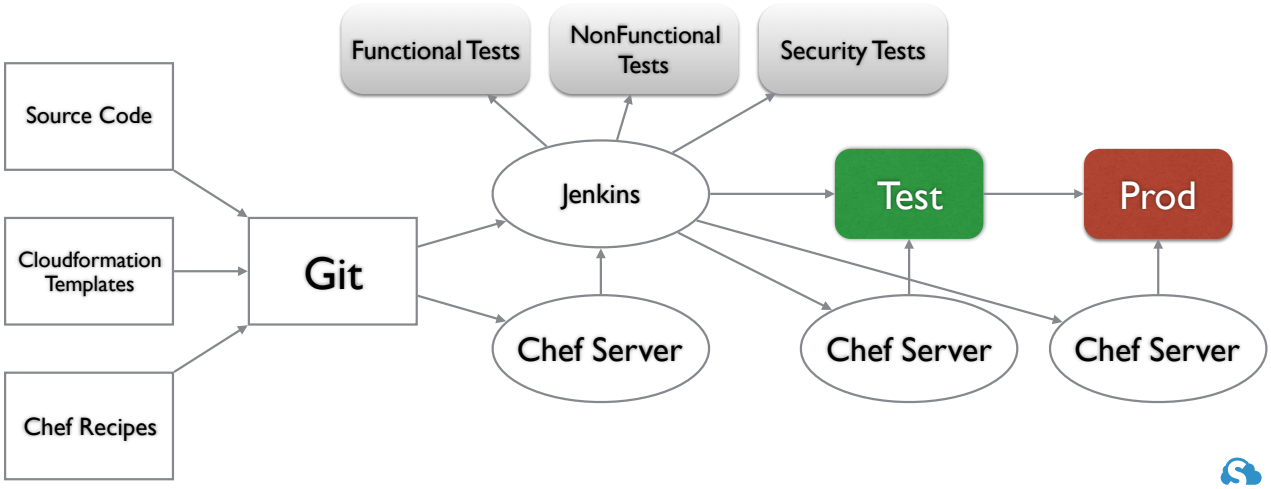
Securosis

Enter DevOps

- DevOps is an *operational framework* that increases standardization, agility, and reliability.
- It relies heavily on virtualization, cloud, and automation.
- The principles come direct from Denning and Lean Manufacturing.

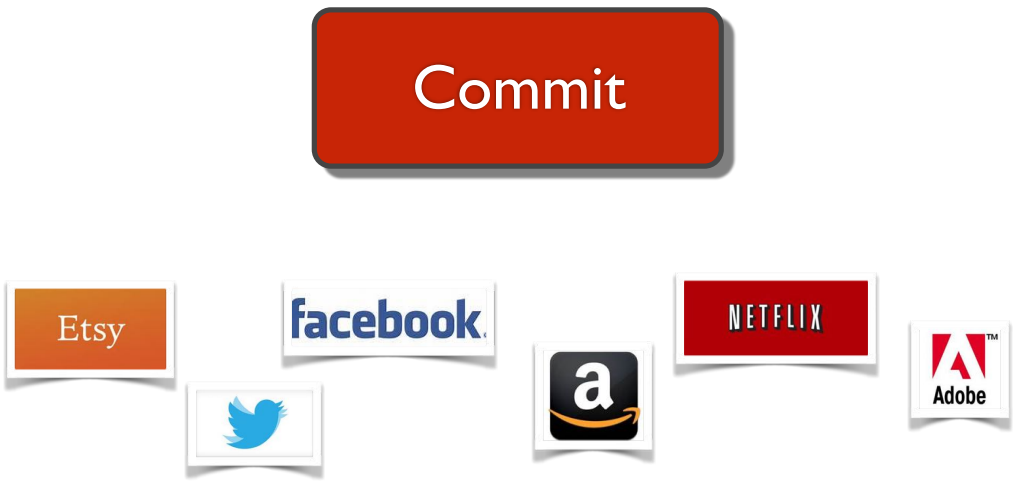
Securosis

Development Pipelines and Continuous Deployment



Securosis

10 Deploys a Day?



Securosis

Why DevOps Works

- All environments, including supporting third-party applications, are consistent.
- The deployment pipeline is automated, for code, configurations, and toolsets.
- There is no drift.There are no human-induced errors.
- Faster deployment cycles reduce error and improve business agility.
- Version control and consistency support instant rollback.

Securosis

[Sec]DevOps

And Rugged DevOps

Securosis

Dev urity

Securosis

Tru  rity

Securosis

DevOps + Security = ♡

- DevOps provides a **consistency** and **control** impossible with manual application deployments.
- Security can easily **embed** and **automate**.
- Security can steal DevOps techniques to apply to diverse workloads and infrastructure requirements.

Securosis

Building Your Toolkit

- Integrate security tests into the deployment pipeline
- Build resilient server configurations
- Embrace Stateless Security
- Automate with Code and APIs

Yes, you need to learn some new skills, but you don't need to become a programmer.

Winning SecDevOps

- Add security to deployment pipelines and automate testing.
  - New tools are here, but we need more…
- Build a library of server hardening scrips and inject into config management.
- Integrate directly into your cloud service for alerting/monitoring.
- Create a library of automation code you can mix and match for different security needs.

Securosis

Instance (Server/Host) Security

Securosis

Image Fundamentals

- Don't use ones you can't validate
  - If you use shared, get the direct image ID from the source
  - Many "certified, pre-owned" available
- Clean before creating
  - Swap space
  - Temp files
  - Embedded keys
  - Other credentials

Securosis

Instance Fundamentals

- Three types
  - Normal/long-running
  - Immutable
  - Automated config management
- Deployment type may change host security controls requirements
- Use encryption to protect snapshots
- New management techniques needed
  - Config management changes
  - Vulnerability assessment changes
  - Patch management changes

Securosis

Lab: Vulnerability Assessment

- Amazon Inspector
  - Agent-based vulnerability assessment
  - Pretty basic, but fully integrated
  - We will use it later for security automation by triggering using SNS
- Setup
  - Inspector is already running in your auto scale group instances
    - See the ansible-baseline.yml file we review later for install details
  - Assuming you tagged the instances, they are ready to be inspected
- Configure Inspector in the AWS Console
  - Create an assessment target to match “Environment: Production”
- Create an assessment template that uses the “Common Vulnerabilities and Exposures” rules package

Cloud-Scale Configuration Management

CHEF

puppet

ANSIBLE

SALTSTACK

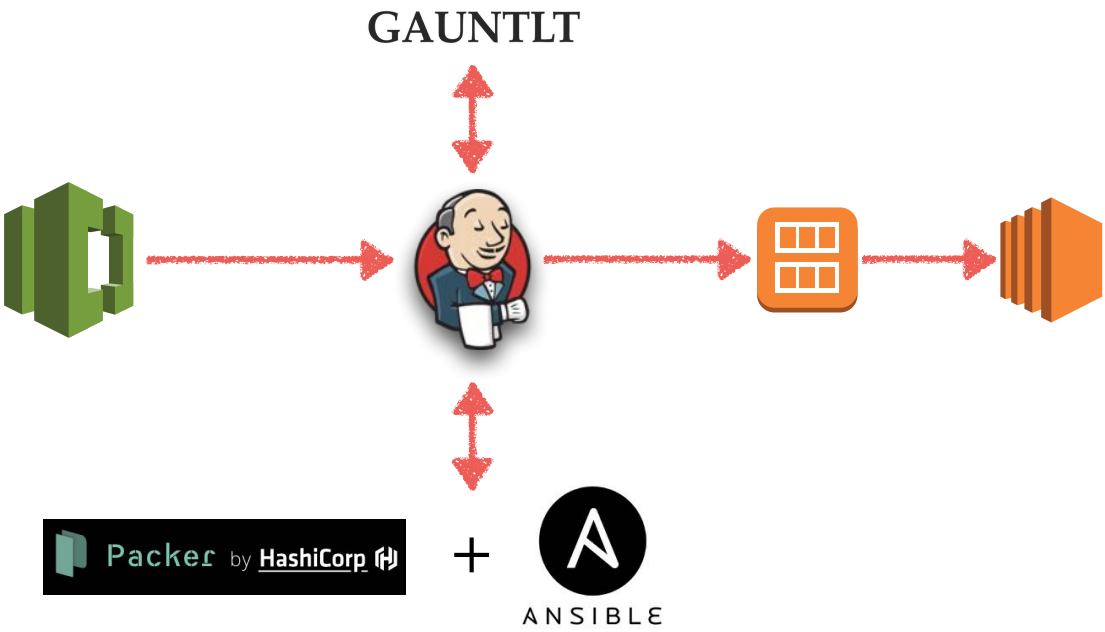
The Power of Immutable

- Instead of updating, you completely replace infrastructure through automation.
- Can apply to a single server, up to an entire application stack.
- Incredibly resilient and secure. Think “servers without logins”.

Image from: http://ourhamptonplaceworld.blogspot.com/2012/07/uluru.html

Securosis

Lab: Building a Deployment Pipeline



Securosis

Before we start, if you get stuck there is a slide at the end of this section with suggestions to solve common problems

Securosis

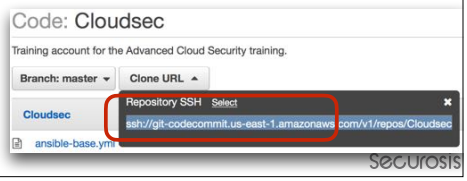
Git and CodeCommit Security

- One of the most popular code repositories
  - GitHub is only one hosting option
  - Works better for most DevOps orgs due to different flow options
  - Self hosted keeps all the data, but lacks visualization and other popular features in GitHub (public or private), Stash, Bitbucket, etc.
  - Always use SSH/TLS options (no self signed certs please), keep repo and OS up to date, and lock it down tight. It's the keys to the code kingdom.
  - Excellent audibility. Copy/store logs externally for audit.
- CodeCommit is AWS's hosted Git
  - Resilient storage on S3
  - Can be fully private and integrate with IAM
  - Can encrypt with KMS

Securosis

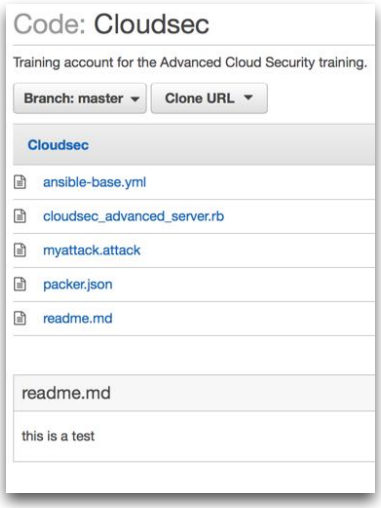
Lab Part 1: CodeCommit

- Start in AWS and create your first CodeCommit repository
  - Create a new IAM user called “Jenkins” and leave the console on that page
  - Create an empty repository called *WebApp[random]*
  - Repository names have to be unique, so figure something out
- Configure your *public Admin instance* with Git for access
  - Double check it has the Dev IAM role assigned
  - Follow the instructions at http://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-ssh-unixes.html
- `git clone <ssh://your repository's URL>`



Lab Part 1: CodeCommit

- Copy the files from the *cloudsec* directory in the class Box folder into the directory you just cloned
  - The URL will be on your cheat sheet
  - Rename the folder *cloudsec*
  - Push the files to your repository
    - `git commit -a`
    - `git push`



Securosis

Lab Part 2: Jenkins

- Connect to your public Admin instance (you should still be)
- Enter these commands (on your cheat sheet):
  - `sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo`
  - `sudo rpm --import http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key`
  - `sudo yum install jenkins`
  - `sudo service jenkins start`
  - `sudo chkconfig --add jenkins`
- Open port 8080 in your security group from your current IP address

Securosis

Lab Part 2: Jenkins

- http://<your instance ip>:8080
- Get the temporary admin password
  - sudo cat /var/lib/jenkins/secrets/initialAdminPassword

Securosis

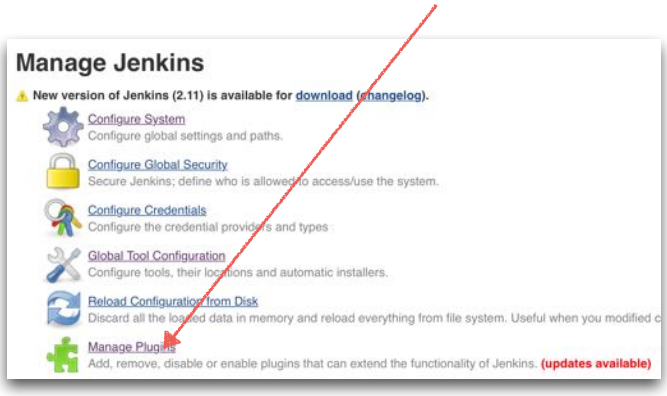
Lab: Install Baseline Jenkins Plugins

- Then create your admin user
  - Pick a super easy username and password
  - 1234 is an excellent option
- Jenkins can run slave nodes, but our entire lab will be on a single master

Securosis

Lab: Add Plugins

- Go to Jenkins > Manage Jenkins > Manage Plugins and install the following plugins:
  - Hudson Post build task
  - Hudson Ruby plugin
  - Packer Plugin
  - Post-Build Script Plugin
  - Parameterized Trigger plugin
  - Ansible plugin
  - Environment Plugin

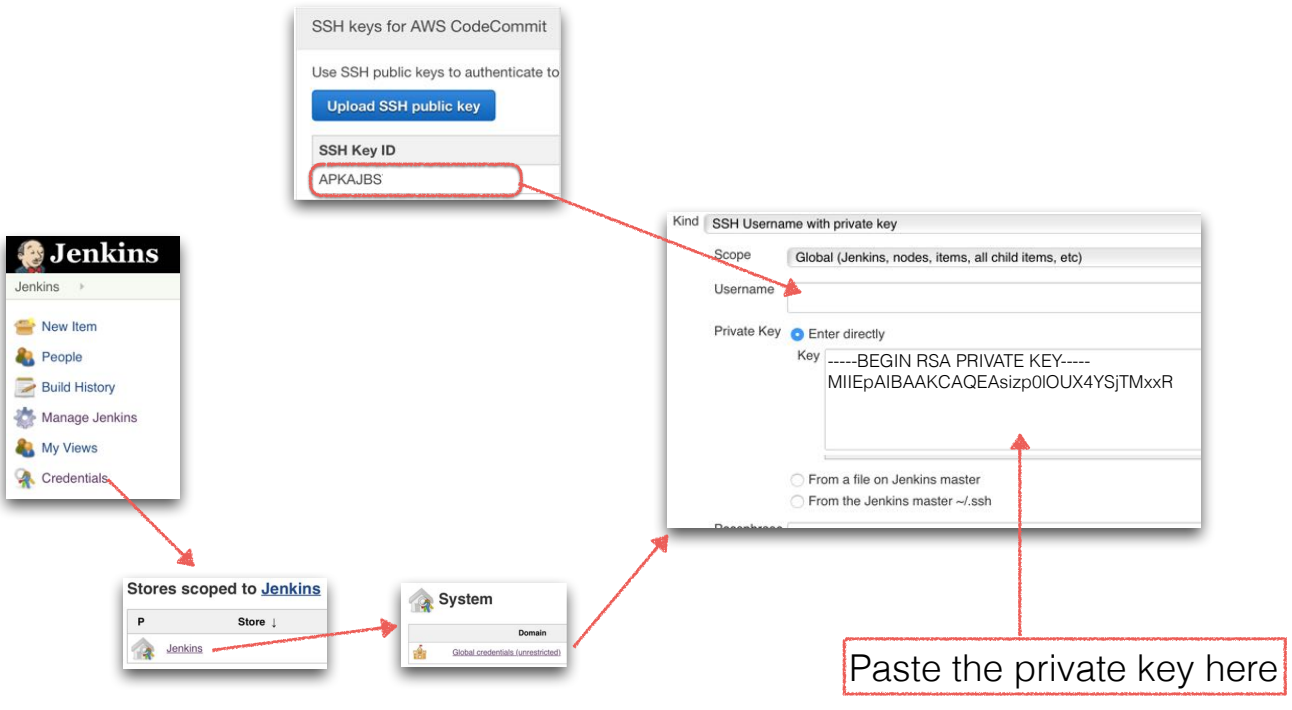


Securosis

Lab: Add Credentials for Git

- Go to *Credentials*
  - Click on the *Jenkins* store > *Global Credentials* > *Add Credentials*
    - The kind is “SSH username with private key”
    - The username is the same you created for CodeCommit (remember, it will look like an Access Key and you find it in your AWS IAM console for that user under SSH Keys
    - Enter the private key directly (cut and paste)

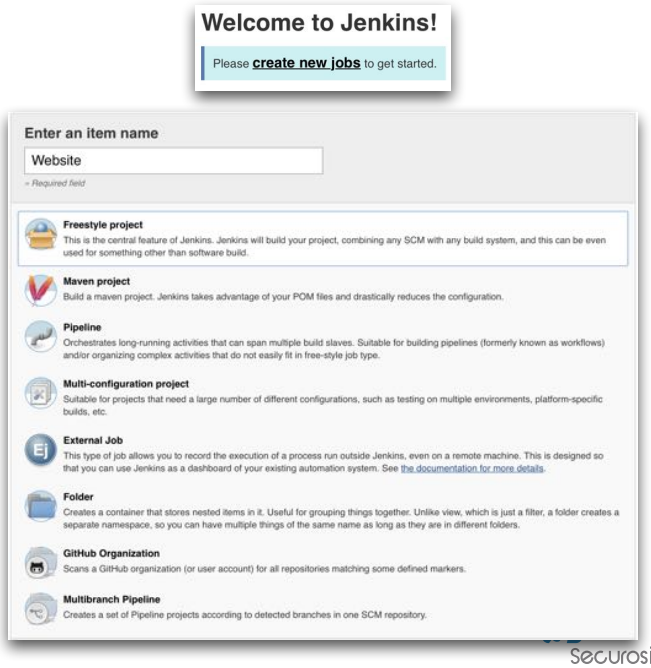
Securosis



Securosis

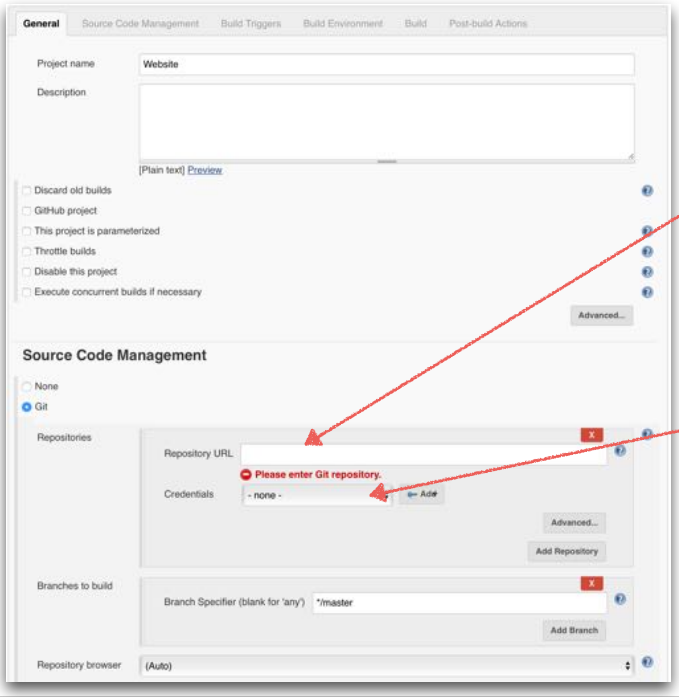
Lab: Create Your First Jenkins Job

- We'll start by pulling from Git and saying Hello World
  - Don't worry, we'll build this out pretty quickly
- Create a new job
  - Name it "Website"
  - Choose a *Freestyle Project*
- Keep in mind, this class focuses on security and we won't be covering the ins and outs of Jenkins



Securosis

Lab: Configure to Pull From CodeCommit



Enter your CodeCommit *ssh url*

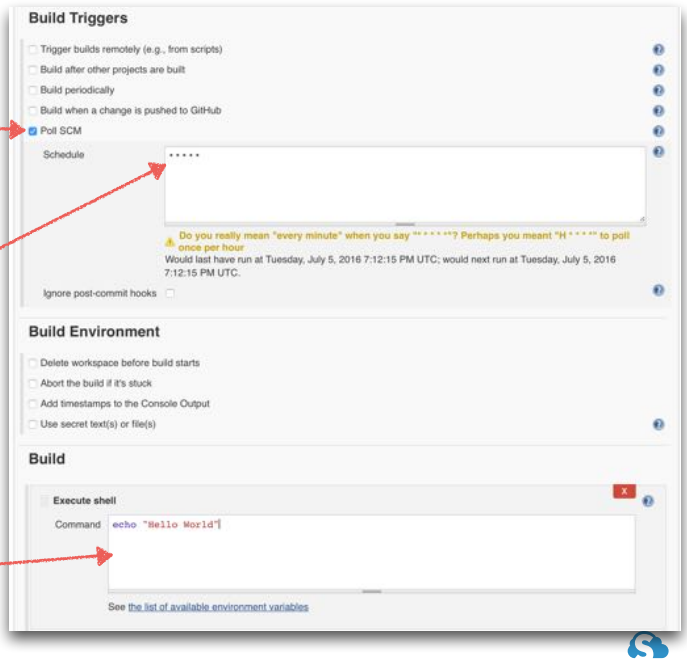
Select the credentials you just configured

Jenkins live-updates so you will know right away if it works

Securosis

Lab: Set the Build Trigger and Hello World

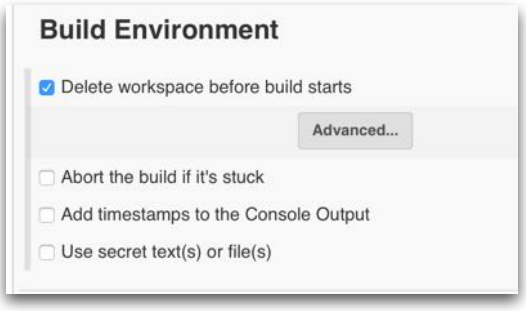
- We will trigger our build on Source Code Management changes (Git)
  - Poll every minute by typing “* * * * *”
  - There’s a space between each *
- *Add Build Step*
  - Our build (for now) will just “Hello World”



Securosis

Lab: Clean Workspace Between Builds

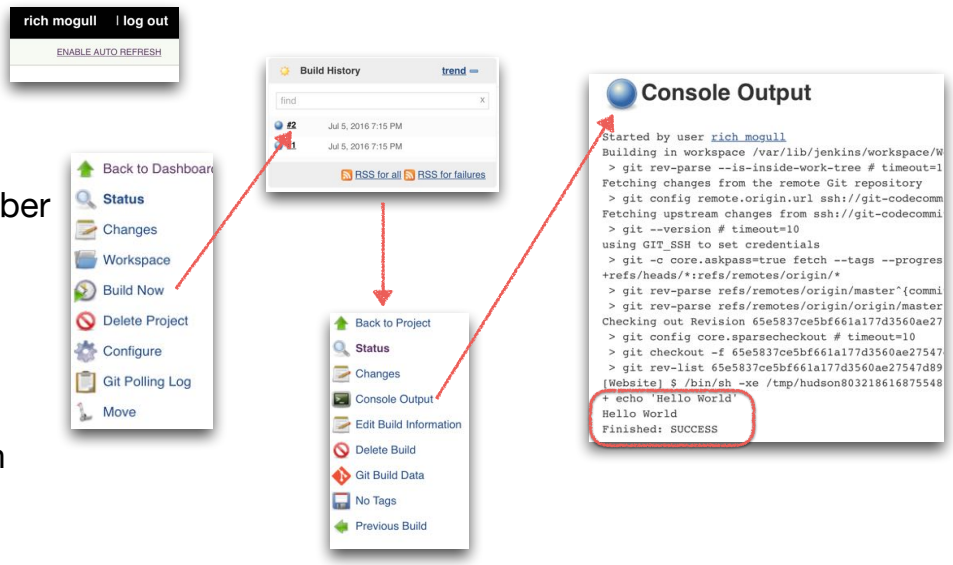
- In large projects this isn’t necessarily a good idea since creating everything from scratch can take a lot of time
  - Ours is small
- We set a file with parameters that needs to be cleaned after every build
  - You could also just swap the file programatically if you want, but this ensures it’s always clean during our experiments



Securosis

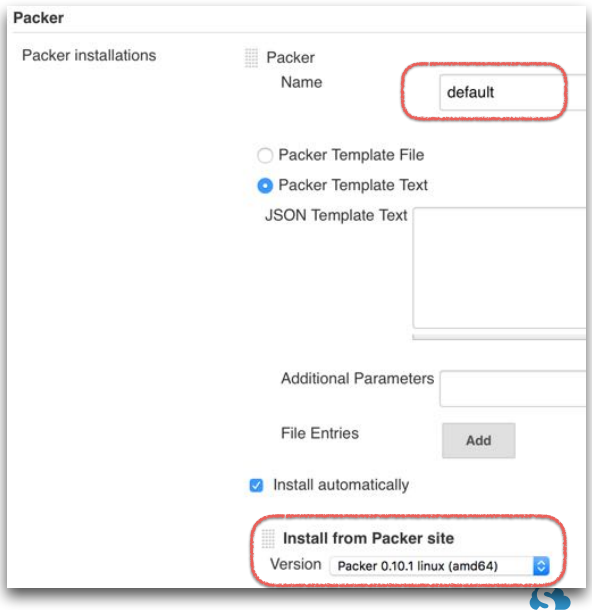
Lab: Save the Job then Test

- Click *Enable Auto Refresh* on the job screen (upper right)
- Click *Build Now*
- Click on the job number
  - Click on *Console Output*
- Or change the readme.md in the repository and watch the build start automatically



Lab: Configure Packer

- Packer builds images for multiple clouds/ platforms based on templates
  - We will use it to build a new, custom AMI when we make code or configuration changes
  - Packer can build for *multiple environments* based on a single template
    - e.g. AWS AMI, Vagrant, Docker
- *Manage Jenkins > Global Tool Configuration > Add packer*
  - Name it “default”
  - Check “Install automatically”
  - Choose the latest linux amd64 version
- Save

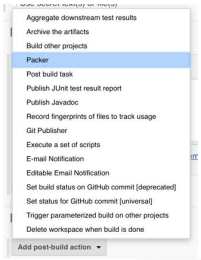


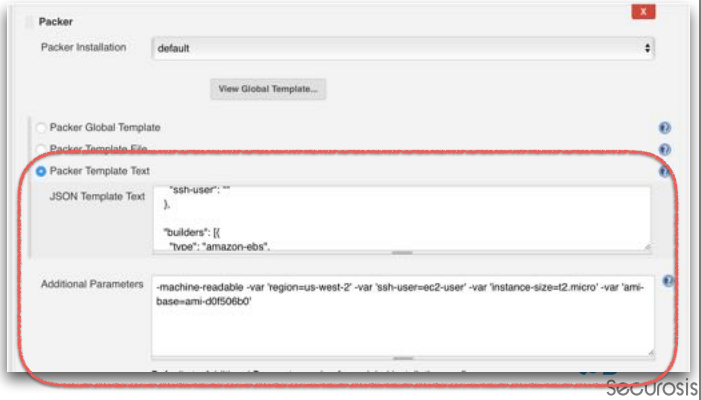
Lab: Configure Packer to Use Ansible

- Packer will create a new AMI based on an existing base
- While it can configure the image itself, we will use Ansible instead since it's more powerful
- Our Packer template will install Ansible locally on the instance, then run Ansible to configure the instance based on a Playbook
  - Ansible runs in local mode
- The Packer template is pasted into Jenkins but the Ansible playbook is pulled in from Git
  - All files in Git are pulled into the job's *workspace*

Lab: Configure Packer to Use Ansible

- Within the job, go to *Configure*
- Add a *Post Build Action*
  - Choose *Packer,* check *Packer Template Text*
  - Paste in the text from the cheat sheet
  - Also paste in the *Additional Parameters*
- Save



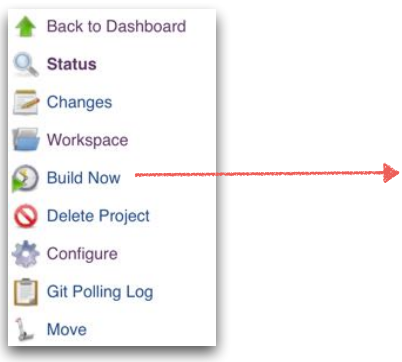


How This Works

- Jenkins polls CodeCommit every minute, and will trigger the build on a change
  - Using SSH credentials
  - In the real world, you would make these read only
- Jenkins runs on an instance with an IAM role, so all the tools will use those credentials
  - The build triggers and echoes “Hello World” to the console. For our lab we aren’t compiling code, so that’s the only build action.
  - Then Packer:
    - Launches a new instance
    - Connects over SSH
    - Installs Ansible
    - Runs Ansible using the playbook
    - Shuts down the instance
    - Creates a new AMI based on the running instance
    - Cleans itself up

Securosis

Run The Build




```
1467750026,,ui,say,==> amazon-ebs: Stopping the source instance...
1467750026,,ui,say,==> amazon-ebs: Waiting for the instance to stop...
1467750083,,ui,say,==> amazon-ebs: Creating the AMI: cloudsec-training-1467749937
1467750083,,ui,message,    amazon-ebs: AMI: ami-d6b676b6
1467750083,,ui,say,==> amazon-ebs: Waiting for AMI to become ready...
1467750134,,ui,say,==> amazon-ebs: Terminating the source AWS instance...
1467750141,,ui,say,==> amazon-ebs: Cleaning up any extra volumes...
1467750141,,ui,say,==> amazon-ebs: No volumes to clean up%!(PACKER_COMMA) skipping
1467750141,,ui,say,==> amazon-ebs: Deleting temporary security group...
1467750141,,ui,say,==> amazon-ebs: Deleting temporary keypair...
1467750141,,ui,say,Build 'amazon-ebs' finished.
1467750141,,ui,say,\n==> Builds finished. The artifacts of successful builds are:
1467750141,amazon-ebs,artifact-count,1
1467750141,amazon-ebs,artifact,0,builder-id,mitchellh.amazonebs
1467750141,amazon-ebs,artifact,0,id,us-west-2:ami-d6b676b6
1467750141,amazon-ebs,artifact,0,string,AMIs were created:\n\nus-west-2: ami-d6b676b6
1467750141,amazon-ebs,artifact,0,files-count,0
1467750141,amazon-ebs,artifact,0,end
1467750141,,ui,say,--> amazon-ebs: AMIs were created:\n\nus-west-2: ami-d6b676b6
Finished: SUCCESS
Finished: SUCCESS
```

Console output with your new AMI

Securosis

Automated Security Testing

- Continuous Integration tools are designed not only to build code, but to automatically test it
- We can leverage this for security testing
  - Security unit tests
  - Code scanning/assessment
    - SAST/DAST/RAST
  - Configuration and environment testing
    - Vulnerability assessment

Securosis

Lab: Security Testing with Gauntlt and Nmap

- Gauntlt automates security tests with adapters included for multiple tools
  - Tests are defined as “Attacks” using human readable language
  - It integrates extremely easily with CI tools
  - You know, like Jenkins
  - Cucumber-compatible

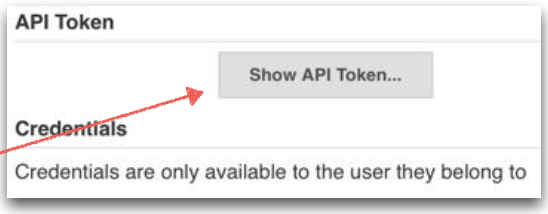
```
# nmap-simple.attack
Feature: simple nmap attack to check for open ports

  Background:
    Given "nmap" is installed
    And the following profile:
      | name     | value       |
      | hostname | example.com |

  Scenario: Check standard web ports
    When I launch an "nmap" attack with:
      """
      nmap -F <hostname>
      """
    Then the output should match /80.tcp\s+open/
    Then the output should not match:
      """
      25\/tcp\s+open
      """
```

Securosis

Lab: Prep and Install Gauntlt

- Create a security group “Nmap Test” and open all traffic from your *Jenkins security group* (copy and paste the sg-***** ID)
- Log into your Jenkins instance:
  - `sudo yum groupinstall "Development Tools"`
  - `sudo yum groupinstall "Development Libraries"`
  - `sudo yum install ruby-devel`
  - `sudo yum install nmap`
- Get your Jenkins API key

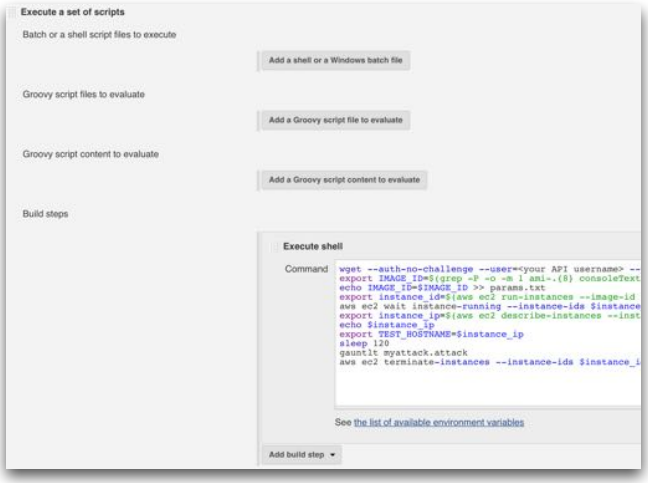
Upper right corner of Jenkins UI



Securosis

Lab: Add the Post Build Script

- Go back into your job’s configuration
- Add a *Post-build action*
  - Choose *Execute a set of scripts*
  - Add a *Build Step*
    - Choose *Execute shell*
    - Paste in the text from the cheat sheet
      - Don’t forget to replace the name and password with your API token credentials
      - Substitute in an AWS host key, the Nmap security group id, and a subnet id (admin public is easiest)
      - If you used a name other than “Website” for the project, you need to swap that into the wget url



Securosis

The Gauntlt Attack File

- Tests to ensure:
  - Port 22 is closed
  - Port 4567 is open

```
@slow
Feature: simple nmap attack

  Background:
    Given "nmap" is installed

    And the following environment variables:
      | name     | environment_variable_name |
      | hostname | TEST_HOSTNAME             |

    And the following profile:
      | name     | value |
      | ssh_port | 22    |

  Scenario: Verify instance has SSH disabled
    When I launch an "nmap" attack with:
      """
      nmap <hostname>
      """
    Then the output should not contain:
      """
      22/tcp
      """
  Scenario: Verify server is open on expected set of ports
    When I launch an "nmap" attack with:
      """
      nmap -p 4567 <hostname>
      """
    Then the output should match:
      """
      4567/tcp\s+open
      """
```

Lab: Fail the Build

- Now fail your build by changing the Ansible Playbook to open up SSH on port 22
  - Comment out these lines (with #) in *ansible-base.yml*
    - `- name: disable ssh`
    - `command: chkconfig off sshd`
    - `command: rpm -e openssh-server`
  - Commit, then push, your changes into your repository
    - `git commit -a`
    - `git push`
  - Watch your console log for the build that starts and see what happens

Play Time

Change the Ansible Playbook and Gauntlt attack files to test out different configurations

Try patching the build and then seeing the results change in AWS Inspector

Securosis

Lab: From Build to Deploy

- We now have a golden image, it's time to deploy it into production
  - This is the core of immutable infrastructure
- We will automatically deploy new images that pass all automated testing via rolling update
  - Despite what you read, not everyone pushes every commit into production
  - We will for demo purposes, but in a real environment you may want deeper testing and manual approvals
  - The techniques we will show will work for even manual deployments

Securosis

Lab: Add Deploy Script to Git

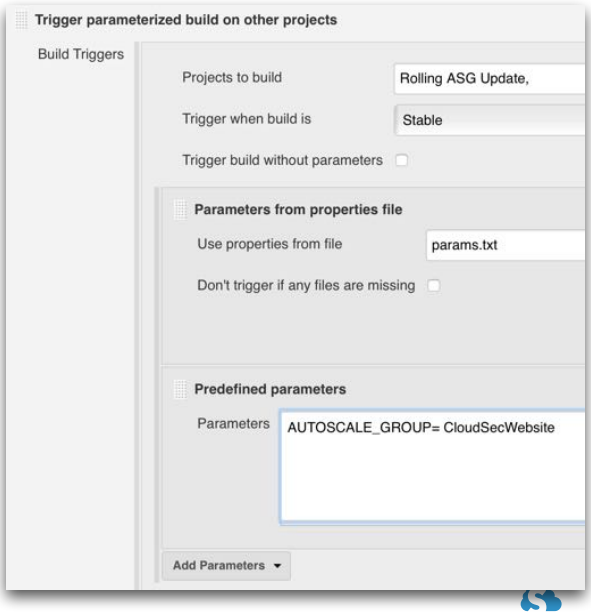
- Create a new CodeCommit repository in the AWS console
  - Or command line if you know how
- Clone it to your Jenkins instance
  - `cd ~`
  - `git clone <url>`
- Add the *rolling_update.rb* script
  - `sudo vim rolling_update.rb`
  - `i` to enter insert mode
  - Copy and paste in the text of the code
  - `<esc>` then `:x`
  - `git add -A`
  - `git push`
- In the AWS console, set your ASG for a Min/Desired instance count of 5

Lab: Create a Second Job

- Name it “Rolling ASG Update”
- Under *General* check *This build is parameterized*
  - Set two string parameters
    - `IMAGE_ID`
    - `AUTOSCALE_GROUP`
  - Under *Build* add an *Execute shell* step
    - `gem install aws-sdk -N`
    - `ruby rolling_update.rb -y $AUTOSCALE_GROUP $IMAGE_ID`

Lab: Trigger the Deploy Job After a Successful Build

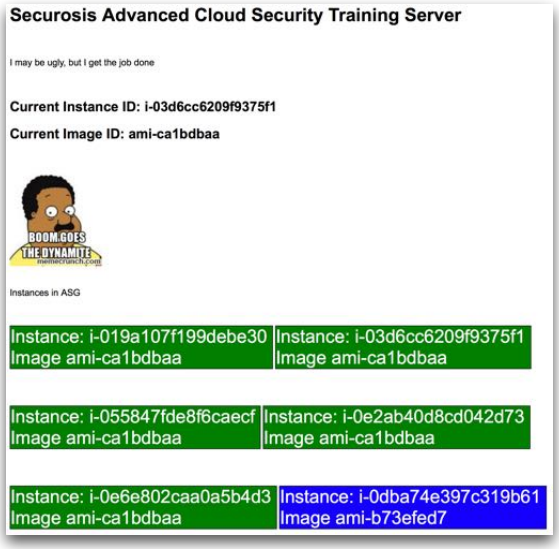
- Go back to your *Website* job
- Add a *Post build action*
  - *Trigger parameterized build on other projects*
  - Enter “Rolling ASG Update” (it will autocomplete)
  - *Add Parameters from properties file*
    - params.txt
  - Add *Predefined parameters*
    - `AUTOSCALE_GROUP= Website`
    - (Or whatever you named your ASG)



Securosis

Lab: Automatic Rolling Updates

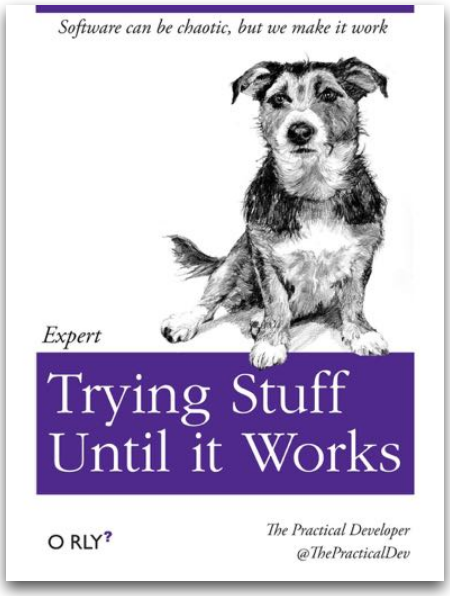
- Manually trigger a *Website* job
  - Assuming it succeeds, it will then trigger the *Rolling ASG Update* job
- Now try a real-world scenario
  - Change Ansible to update vulnerabilities detected by Inspector
  - Commit to Git, and watch the build automatically start and roll out
  - Instance colors will change as you have a mix of AMIs



Securosis

Problems? Potential Fixes

- There are a bunch of settings you need to substitute in the scripts (e.g. security group ID, subnet, etc.). Make sure those are all correct.
- If your subnet isn't set to provide a public IP address, GauntIt will fail based on how things are configured. It's an easy fix from *VPC* > *subnets* under A*ctions.*
- If you manually changed the auto scale group settings something may have gotten messed up.
  - Or if the script partially ran, settings may be in an in-between state.
  - In both cases, an easy fix is to swap back to the original launch configuration, then detach/ terminate the instances currently in the group.
- Make sure your job is set to clean the workspace before (or after) each run.



Securosis

Software Defined Security

Securosis

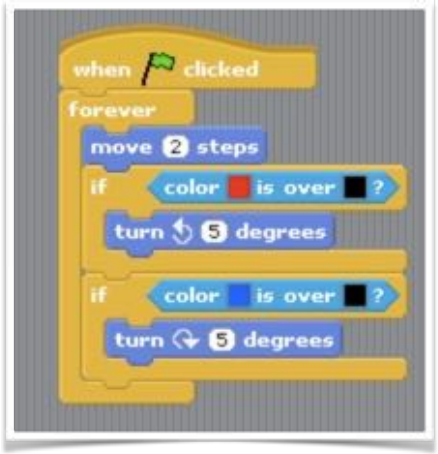
Software Defined Security

- Attackers are automated, we are mostly manual.
- Our tools have been poor.
- We lack trustable security automation and thus need to rely on a “Meat Cloud”
- In cloud, APIs are mandatory. We can write code to automate and orchestrate, even across products and services.

Securosis

Code without coding

- Work with your devs to build a library of building blocks
- Learn just enough to glue it together
- Build some core scrips
- Mix and match the blocks



Securosis

Demo

- Meet SecuritySquirrel, the first warrior in the Rodent Army (apologies to Netflix).
- The following tools are written by an analyst with a Ruby-for-Dummies book.
- Automated security workflows spanning products and services.

Securosis

Incident Response

1. Detect Compromise
2. Pull server information (If you have it)
3. Quarantine
4. Imag
5. Analyze
6. Recover

= Hours!

Each step is manual, and uses a different set of disconnected tools

Securosis

1. Pull metadata
2. Quarantine
3. Swap control to security team
4. Identify and image all storage
5. Launch and configure analysis server
6. Can re-launch clean server instantly

Securosis

Stateless Security

- Security normally relies on scanning and checking databases.
- With cloud we are completely integrated into the infrastructure and platforms.
  - The cloud controllers have to see everything to manage everything, there is no Neo running around.
- Instead of scanning, we can directly pull state.
  - And then use it for security

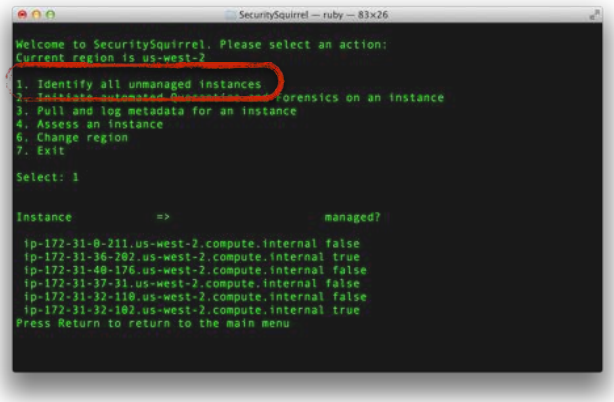
Securosis

Identify Unmanaged Servers (for the audit)

1. Scan the network
2. Scan again and again for all the parts you missed
3. Identify all the servers as best you can
4. Pull a config mgmt report
5. Manually compare results

Securosis

DEMO



- Get list of all servers from cloud controller (can filter on tags/ OS/etc).
  - Single API call
- Get list of all servers from Chef
  - Single API call
- Compare in code

Securosis

Code Walk Through

- Key aspects:
  - Authentication/authorization via Roles
    - Initializing clients
  - Understanding method and variable scope
  - AWS SDK/JSON navigation
    - Structs > hash > arrays
    - Hidden complexities (e.g. ENIs and security groups)
  - Tips
    - Waiters
    - Managing API limits
    - CLI vs. SDK (—query)

Securosis

Lab: Software Defined Security

- We've provided a library of code to start
- Define a security problem
  - Incident Response
  - Finding security holes
  - Enabling security settings
- Break into teams
- Start programming

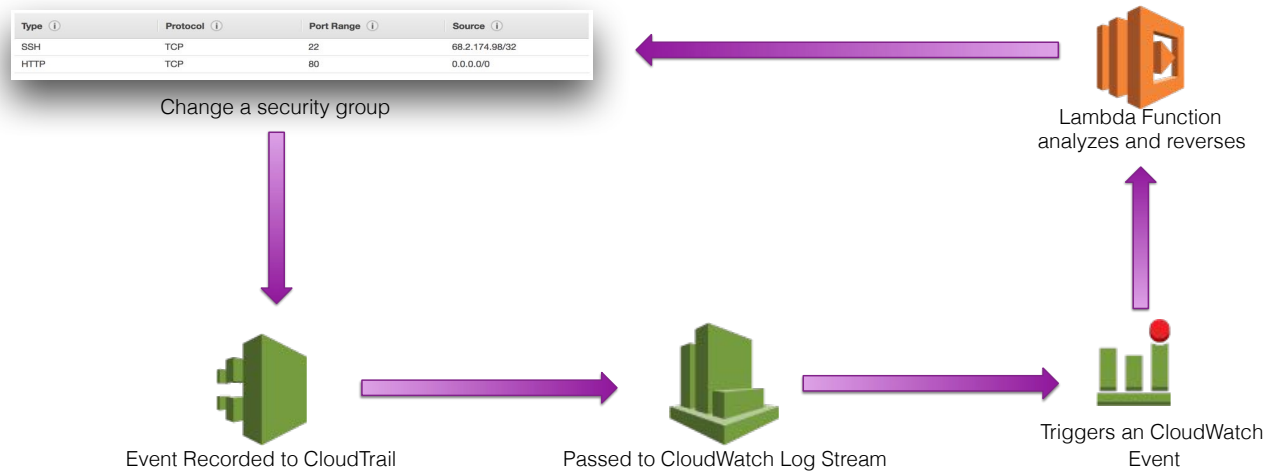
Securosis

Event Driven Security

- Cloud providers are creating hooks to trigger actions based on events inside the cloud.
- We can use these for near-instant security reactions.

Securosis

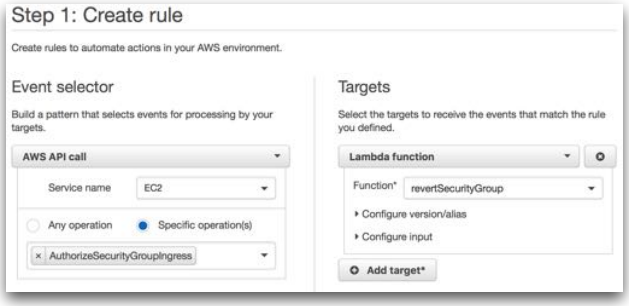
Self-Healing Infrastructure (yes, for real)



Securosis

Lab: Event Driven Security

- Create a new Lambda function
- Paste in the provided code to revert security group changes
  - Alter it to trigger based on a conditional
- Create a new CloudWatch Rule
  - Service Name: EC2
  - Specific Operations: AuthorizeSecurityGroupIngress
  - Target your Lambda function
    - Add an SNS notification if you want
- Test it
- Free time? Write/edit your own



Securosis

Reminder: We are not responsible for your AWS bill if you forget to delete things after class finishes.

Really… costs are minimal for our 2 days, but will add up if you leave things running.

Securosis