CloudFormation Deep Dive

AWS CLI Install & Config

AWS Course Author: Craig Arcuri

Details on the CLI

- Open source tool built on top of the AWS SDK for Python (Boto) that provides commands for interacting with AWS services.
- Available terminal programs:
    - Linux Shells
    - Windows Command Line
    - Remotely (PuTTY or SSH)
- Direct access to AWS services' public APIs.

Installing the CLI

- The primary distribution method for the AWS CLI on Linux, Windows, and macOS is pip (python package manager).
- Pip provides an easy way to install, upgrade, and remove Python packages and their dependencies.
- If you already have pip and a supported version of python, you can install the AWS CLI with the following command:

  $pip install awscli –upgrade –user

- --upgrade tells pip to upgrade any requirements that are already installed.
- --user tells pip to install the program to a subdirectory of your user directory to avoid modifying libraries used by your operating system.
- After you install the AWS CLI, you may need to add the path to the executable file to your PATH variable.

Verify the CLI Installation

- Verify that the AWS CLI installed correctly by running:

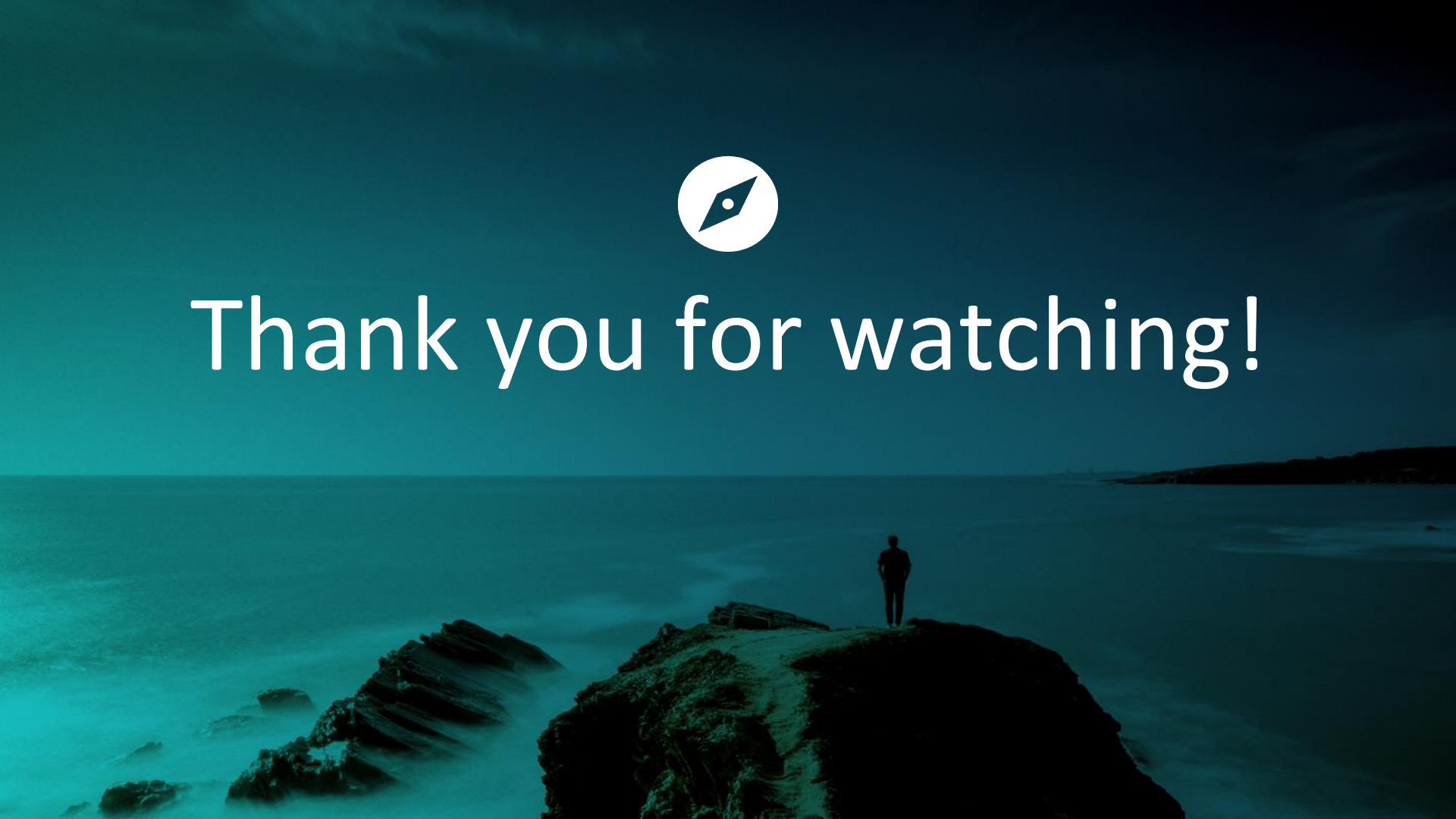
  aws –version

- The AWS CLI is updated regularly to add support for new services and commands. To update to the latest version of the AWS CLI, run the installation command again.

  $ pip install awscli –upgrade –user

- If you need to uninstall the cli use:

  $ pip uninstall awscli


Thank you for watching!

CloudFormation Deep Dive

Using AWS Config to Monitor Stacks

AWS Course Author: Craig Arcuri

AWS Config supports stacks!

- Track the current and historical configuration of your CloudFormation stacks.
- Get SNS notifications when your configuration changes.
- Use a managed AWS Config rule to check whether your CloudFormation stacks are sending event notifications to an SNS topic.
- Great for troubleshooting and for audit compliance.
- View stack changes inside the AWS Config timelie.

AWS Config

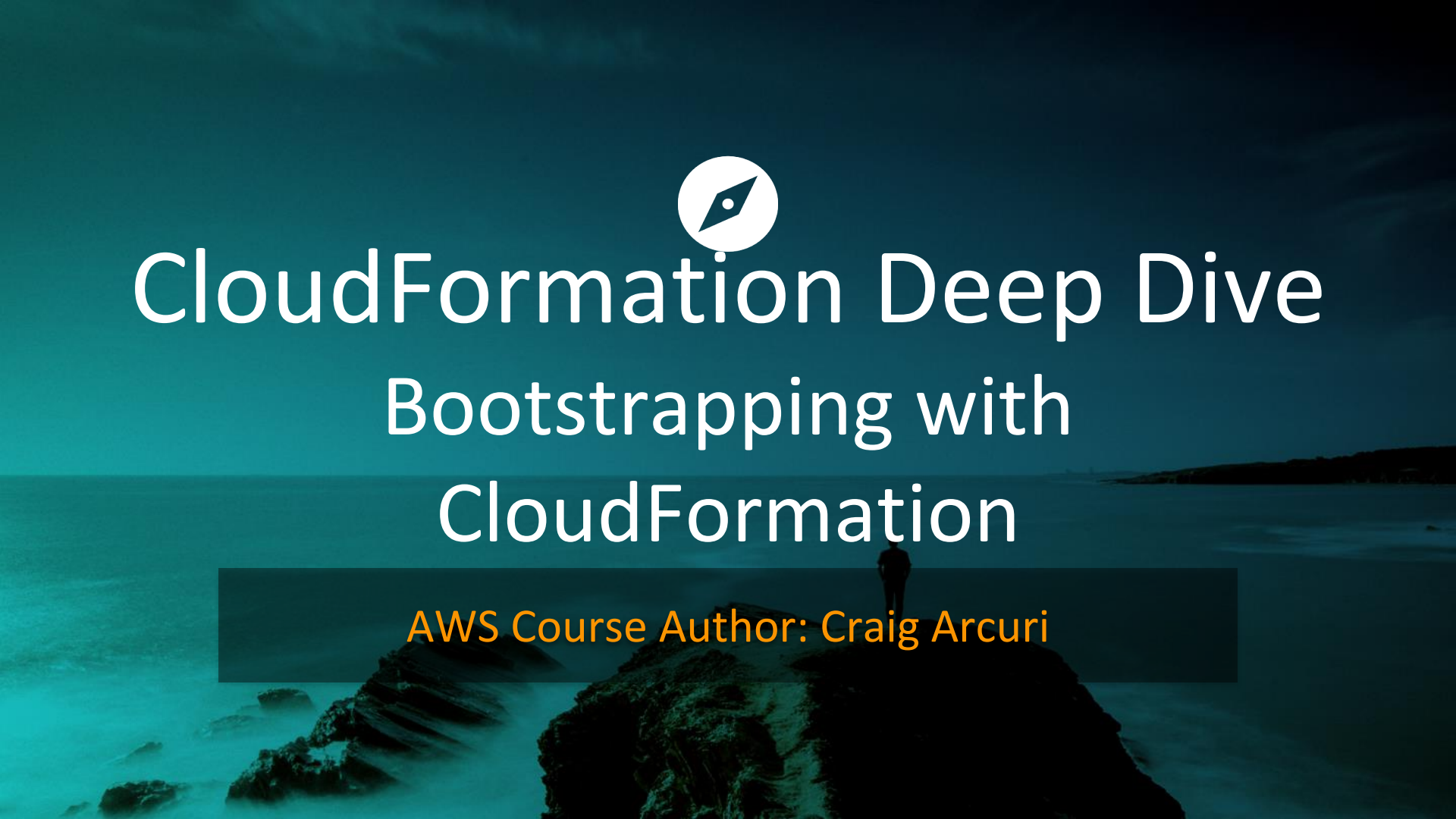
- AWS Config enables you to audit and assess the configurations of your AWS resources.
- Monitors and records your resource
- Automate the evaluation of recorded configurations against desired configurations.

Track Your Stack

- You have created a stack in production.
- You create a stack policy of deny all.
- Another administrator replaces the stack policy with a policy that Allows all.
- He then makes changes to the stack.
- With AWS Config and SNS notifications, you are notified of these changes.

AWS Config Rules

- Use AWS Config rules to check your stacks against best practice rules and internal policies.
- Pre-built rues: **cloudformation-stack-notification-check** - verify whether your CloudFormation stacks are sending event notifications to an SNS topic at all times.
- Can show compliant and noncompliant configuration states.
- Rules represent your desired configuration setting.
- AWS Config evaluates whether your resource configurations comply with relevant rules and summarizes results in a table.



CloudFormation Deep Dive

Bootstrapping with CloudFormation

AWS Course Author: Craig Arcuri

Bootstrapping Applications

- CloudFormation takes care of provisioning your resource, but you must still deploy your applications.
- How can you deploy your applications during CloudFormation resource provisioning? Bootstrapping!

Different bootstrapping options:

- Use CloudInit and EC2 user data to customize an instance at launch time.
- Use CloudFormation Metadata and Helper scripts.
- Bake your apps into an AMI
- Integrate with Chef or Puppet

Using CloudInit and User Data

- With CloudInit you can pass executable actions to instances at launch via user-data fields.
- Ubuntu and Amazon Linux AMIs contain a version of CloudInit.
- Can use a common base AMI due to the ability to dynamically configure an AMI at startup.
- Download and install software without having to have applications preinstalled in the AMI.
- Can also use a hybrid approach where some of the core applications can be built in to the AMI.

Customize instances with CloudInit and User Data

- Pass information to the AMI to customize at launch time.
- Do this using the user data field within the metadata store.
- Use Intrinsic Function (Fn::Base64, Fn::GetAtt, Fn::Join, Ref) to construct strings which get passed to EC2 user data.
- The string format is typically value pairs.

User Data Example

```
"MyInstance": {
"Type": "AWS::EC2::Instance",
  "Properties": {
    :    "UserData": {
     "Fn::Base64": { "Fn::Join": [ "", [
      "Database=", {"Fn::GetAtt": ["SampleDB", "Endpoint.Address"]}, ":", {"Ref": "DatabasePort"}, "&",
      "DBUser=", {"Ref": "DatabaseUser"}, "&",
      "DBPassword=", {"Ref": "DatabasePassword"}, "&",
      "WSPort=", {"Ref": "WebServerPort"}
     ]]}}
         }
     }
 }
```

- The instance needs to have a script that retrieves the string from the EC2 metadata store and acts on it.
- A helper script can also be used to extract the user data.

Working with User Data

- For many Linux AMIs add your scripts to the /etc/rc.local boot script.
- User data is immutable and can not be changed after an instance is launched.
- You can instead pass the location of data (such as an S3 bucket) through the user data rather than the data itself.

Using Metadata and Helper Scripts

- CloudFormation templates can be used to define the packages, files, and OS services that are installed on EC2 instances. Helper scripts run on the instance to interpret the definitions in your template, installing packages, creating files, and starting services on the instance.
- Helper scripts build on CloudInit functionality allowing you to create a common CloudInit startup script driven from the template.
- You describe the applications to be installed and CloudFormation takes care of the how.

Using Metadata and Helper Scripts

- You can attach metadata to any resource.
- Intrinsic functions in the metadata are resolved at run time.
- Applications can retrieve metadata using the CLI command get-stack-resource or the API call GetStackResource.

Baking Applications into an AMI

- Use AWS AMIs or published custom AMIs which have pre-baked applications.
- Otherwise you can create your own custom AMIs pre-baked with your applications.
- You can access the EC2 instance metadata store to further customize AMIs during launch. Example: passing the database connection string to the EC2 instance from the metadata store.

Integration with Chef and Puppet

- Chef is an open source infrastructure automation solution written in Ruby.
- Chef automates the configuration of your systems and the apps on top of it.
- It is a client-server app where clients pull configuration from the Chef server.
- Puppet is an Open Source IT services platform.
- With Puppet, you can manage provisioning, patching, and configuration of OS and their app stacks.
- Puppet is a client-server app where clients pull configuration from a puppet master.
- With CloudFormation you can bootstrap your EC2 instances with the Chef or Puppet client software. You can also deploy and configure a Chef server or a puppet master.

CloudFormation Deep Dive Change Sets

AWS Course Author: Craig Arcuri

Using Change Sets

- Change sets allow a preview of how proposed changes will impact your running resources.
- Create and manage change sets using the CloudFormation console, CLI, or CloudFormation API.
- However, Change sets don't indicate whether AWS CloudFormation will successfully update a stack.
- Steps:

1 Create a Change Set

2 View the Change Set (preview changes)

3 Create additional Change Sets (optional)

4 Execute the Change Set

Creating a Change Set

- Create a change set for a running stack:

Submit the changes that you want to make by providing a modified template, new input parameter values, or both.

- CloudFormation generates a change set by comparing the stack with the changes you submitted.

Viewing a Change Set (splitscreen)

- After creating a change set, view the proposed changes before executing them.
- Use CloudFormation console, CLI, CloudFormation API to view change sets.
- Console provides a summary of the changes list of changes in JSON format.

Executing a Change Set (splitscreen)

- Execute the change set after reviewing the changes.
- After executing a Change Set, CloudFormation deletes all change sets associated with the stack because they are no longer valid for the updated stack.
- To update a protected resource with a Change Set, you must first update the stack policy.
- Steps:

1 Choose the stack to update in the console

2 Choose ‘Change Set’s in the Detail Panel

3 Choose the Change Set to execute

4 Choose Execute

5 Confirm the Change Set, then execute

Deleting a Change Set (splitscreen)

- Deleting a change set removes it from the list of change sets for the selected stack.
- Steps:

1 Choose the stack that contains the change set that you want to delete.

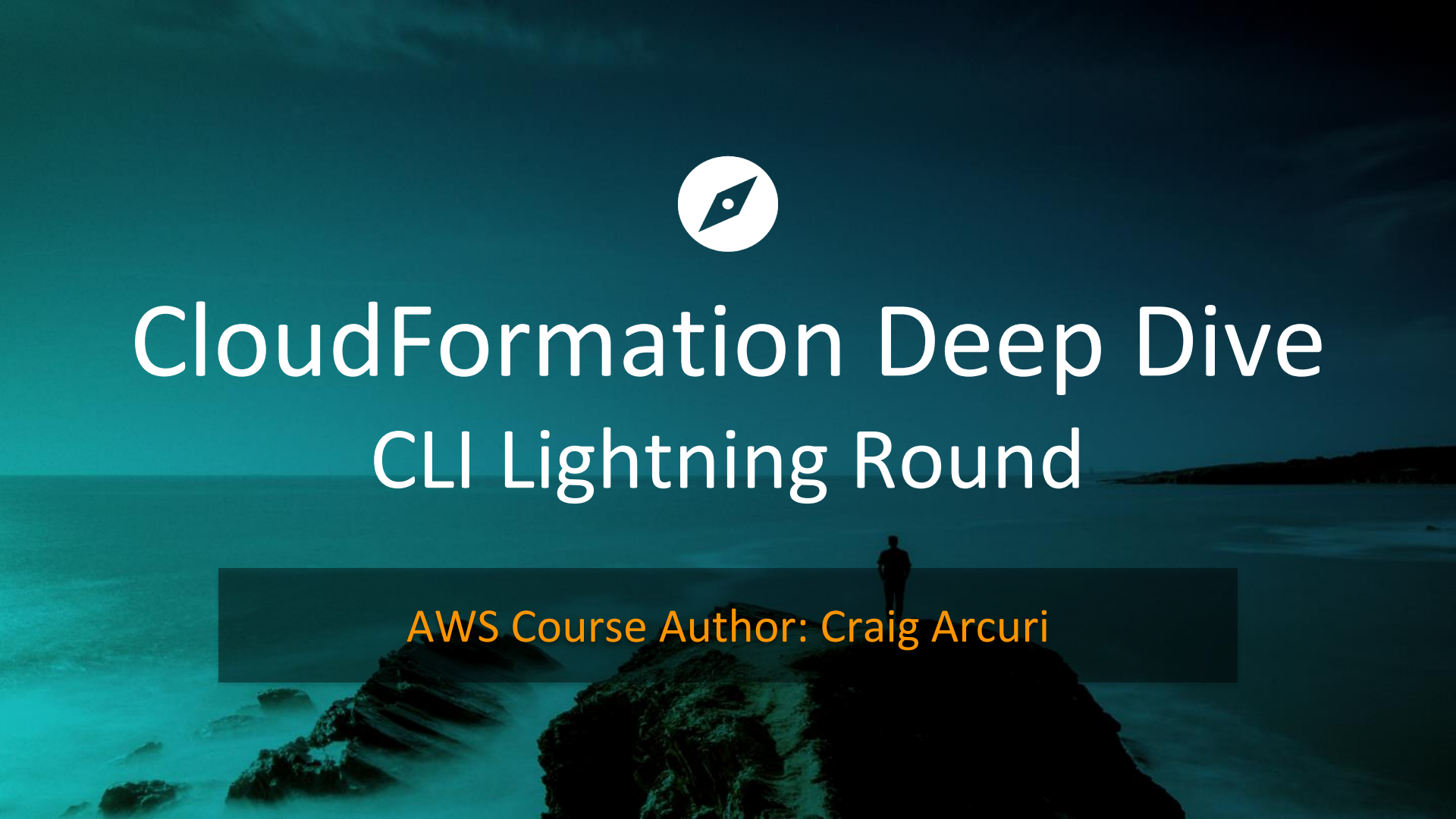
2 Choose **Change Sets** in the stack Detail Panel.

3 Choose the change set that you want delete.

4 Choose **Other Actions**, and then choose **Delete**.

5 Confirm that this is the change set you want to delete, and then choose **Delete**.


Thank you for watching!



CloudFormation Deep Dive

CLI Lightning Round

AWS Course Author: Craig Arcuri

CloudFormation CLI Command Reference

- https://docs.aws.amazon.com/cli/latest/reference/cloudformation/

CloudFormation CLI Lightning Round

- Can you cancel a stack update from the command line?

CloudFormation CLI Lightning Round

- Can you cancel a stack update from the command line?
- Yes, with the cancel-update-stack command
- Cancels an update on the specified stack. If the call completes successfully, the stack rolls back the update and reverts to the previous stack configuration.

CloudFormation CLI Lightning Round

- Can you work with Change Sets from the command line?

CloudFormation CLI Lightning Round

- Can you work with Change Sets from the command line?

- Yes! With the Commands:
- create-change-set
- delete-change-set
- describe-change-set
- execute-change-set
- list-change-sets

CloudFormation CLI Lightning Round

- Which Change Set command returns the inputs for the change set and a list of changes that CloudFormation will make if you execute the change set?

CloudFormation CLI Lightning Round

- Which Change Set command returns the inputs for the change set and a list of changes that CloudFormation will make if you execute the change set?

- describe-change-set

CloudFormation CLI Lightning Round

- Can you work with Stack Instances from the CLI?

CloudFormation CLI Lightning Round

- Can you work with Stack Instances from the CLI?
- Yes!

CloudFormation CLI Lightning Round

- What are the CLI commands to work with Stack Instances?

CloudFormation CLI Lightning Round

- What are the CLI commands to work with Stack Instances?

- create-stack-instances
- delete-stack instances
- list-stack-instances
- update-stack-instances

CloudFormation CLI Lightning Round

- Is there a command to estimate costs of a template from the command line?

CloudFormation CLI Lightning Round

- Is there a command to estimate costs of a template from the command line?
- Yes! estimate-template-cost

CloudFormation CLI Lightning Round

- What is the command to retrieve a stack policy?

CloudFormation CLI Lightning Round

- What is the command to retrieve a stack policy?

- get-stack-policy

CloudFormation CLI Lightning Round

- How can you assign a stack policy to a stack from the cli?

CloudFormation CLI Lightning Round

- How can you assign a stack policy to a stack from the cli?
- set-stack-policy

CloudFormation CLI Lightning Round

- Which commands allow you to work with stack sets?

CloudFormation CLI Lightning Round

- Which commands allow you to work with stack sets?

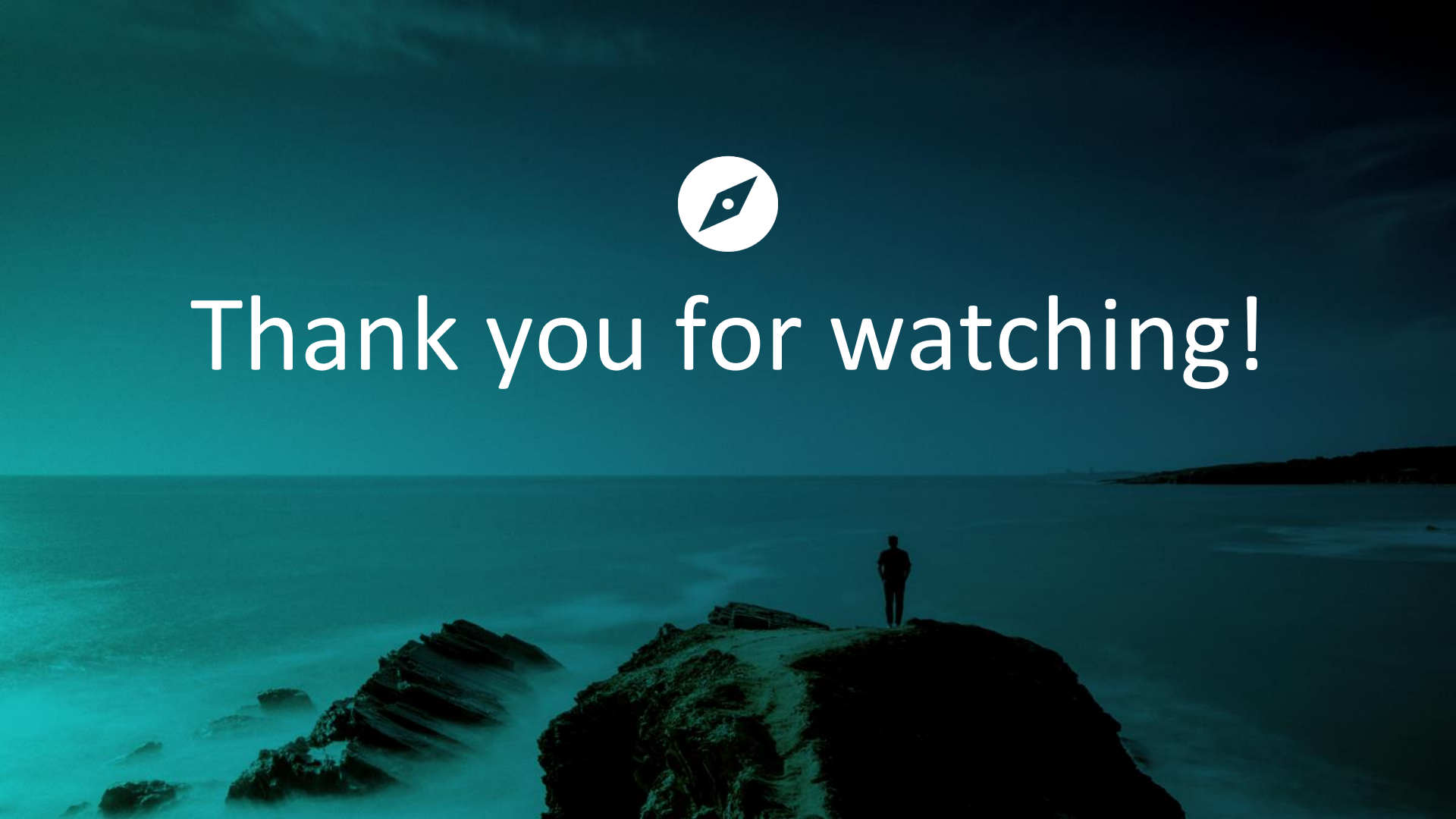
- create-stack-set
- delete-stack-set
- describe-stack-set
- list-stack-sets
- update-stack-sets

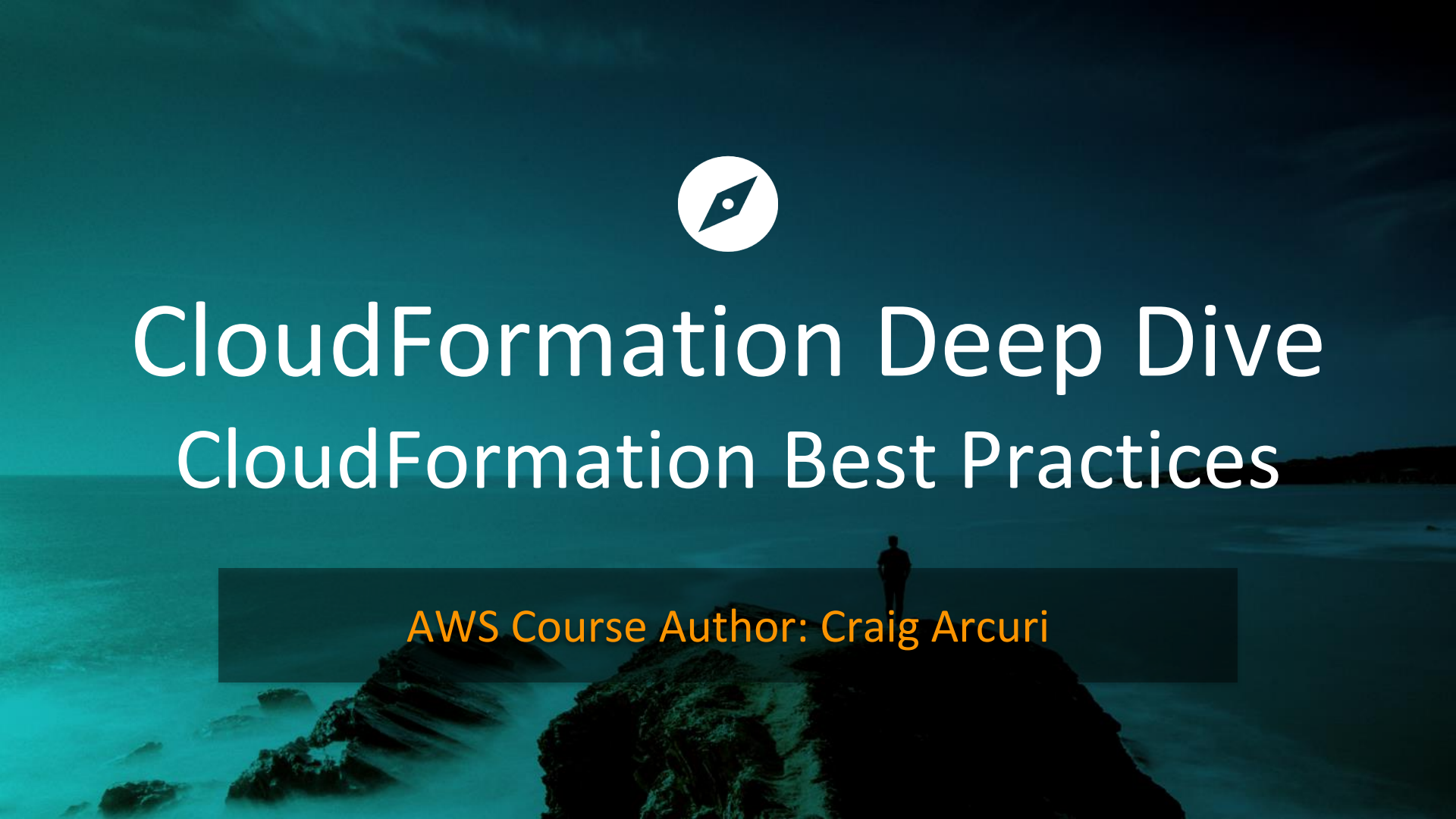
CloudFormation CLI Lightning Round

- How could you review the values being used in cross-stack references?

CloudFormation CLI Lightning Round

- How could you review the values being used in cross-stack references?
- Two commands:
- list-exports
- list-imports


Thank you for watching!



CloudFormation Deep Dive

CloudFormation Best Practices

AWS Course Author: Craig Arcuri

Organize Your Stacks By Lifecycle and Ownership

- Organize stacks by lifecycle and ownership.
- Stacks can grow quickly and management can become messy.
- Group resources with common lifecycles and ownership into separate stacks. Changes can be made to these separate stacks without affecting other stack.
- Two common frameworks for organizing stacks:
    - Multi-layered architecture - organizes stacks into multiple horizontal layers that build on top of one another, where each layer has a dependency on the layer directly below it. Within each layer, stacks should have similar lifecycle and ownership.
    - Service-oriented architecture - Organize business problems into manageable parts. Each opart is a service that has a clearly defined purpose and represents a self-contained unit of functionality. You can map these services to a stack, where each stack has its own lifecycle and owners. All of these services (stacks) can be wired together so that they can interact with one another.

Use Cross-Stack References

- Separating stacks by lifecycle and ownership dictates that you may want to have access to resources in another stack.
- Use cross-stack references to export resources from a stack so that other stacks can use them.

Use IAM To Control Access

- Use IAM to specify CloudFormation actions users can perform.
- Use needs access to CloudFormation but also to specific resources in a stack (S3, EC2, VPC, etc).
- CloudFormation makes calls to create, update, and delete resources in the stack so assign access accordingly.
- Use a service role to separate permissions between a user and a service.

Verify Quotas For Services

- Check you AWS account limits for each service you want to create in a stack. Ensure that you can create each service without reaching your quota.
- Your stack will not be created if you hit a limit for one of your resources.

Reuse Templates

- Reuse templates to replicate your infrastructure in multiple environments.
- Create environments for dev, test, and prod so that you can test changes before implementing them into production.
- Use parameters, mappings, and conditions sections to make templates reusable.

Use Nested Stacks

- Reuse common template patterns with Nested Stacks
- Separate out common components and create dedicated templates for them.

Do Not Hard Code Credentials

- Use input parameters to pass in information whenever you create or update a stack.
- Use noEcho to obfuscate sensitive parameters.

Use AWS Specific Parameter Types

- Use AWS specific parameter type when your template requires inputs for existing AWS-specific values (VPC IDs, EC2 key pair name, etc.)
- Example: AWS::EC2::KeyPair::KeyName
- CloudFormation can quickly validate values for AWS-specific parameter types before stack creation.
- CloudFormation Management Console will show a dropdown list of valid values.

Use Parameter Constraints

- Always good practice to use constraints on input pages to disallow invalid inputs. Validate your parameters!
- Parameter constraints can be used for this purpose.
- Min and Max values on parameters.
- Minimum length and makeup (alphanumeric) on user names.

Use AWS::CloudFormation::Init

- Deploy software on EC2 instances with AWS::CloudFormation::Init and cfn-init helper script.
- Describe the configurations you want instead of scripting procedural steps.
- Can update configurations without recreating instances.

Keep Helper Scripts Up To Date

- Helper scripts are updated periodically.
- yum install -y aws-cfn-bootstrap ← Include in the user data property of your template before calling helper scripts. This will deploy the latest helper scripts to your instances.

Validate Templates

- Validate templates before using them.
- Validation is done automatically in the console.
- CloudFormation Designer has a validate button.
- Validation can be done from the command line or API (validate-template command).
- CloudFormation first checks if the template is valid JSON.
- Then CloudFormation checks if the template is valid YAML.
- Can help you catch syntax and some semantic errors, such as circular dependencies, CloudFormation creates any resources.

Consistently Manage Stack Resources

- Manage all your stack resources through CloudFormation.
- Use the console, CLI, or API to manage your stacks.
- Do not make changes to your stacks outside of CloudFormation. This can create a mismatch between your stack's template and the current state of your stack resources. This is known as Configuration Drift.
- Configuration Drift can cause errors if you update or delete the stack.

More On Configuration Drift

- Configuration Drift = Difference between expected and current values in your stack. These difference arise from making changes to the stack outside of CloudFormation.
- Can cause stack updates to fail.
- Can put the stack in a state where you can't update or delete the stack.
- Creates divergence from your approved architecture.
- Planned Changes Can be tested.
- Changes made to stacks outside of CloudFormation are not always tested. Can cause issues in the stack as well as Configuration Drift.
- Types of modification to: stack resource property values, default values of stack resources, deleting stack resources.
- Can use AWS Config with CloudFormation to combat and monitor this. AWS announced a new feature coming in 2018 – Drift Detection.

Use Change Sets When Updating Stacks

- Allow you to preview proposed changes to a stack and the impact they may have on your running resources before you implement them.
- CloudFormation doesn't make any changes until you execute the change set, allowing you to decide whether to proceed with the changes or create another change set.
- Use change sets to check how changes might affect running resources.

Use Stack Policies

- Stack policies protect critical stack resources from unintentional updates that could cause resources to be interrupted or even replaced.
- Always use a stack policy for stacks stack that have critical resources.

Use CloudTrail With CloudFromation

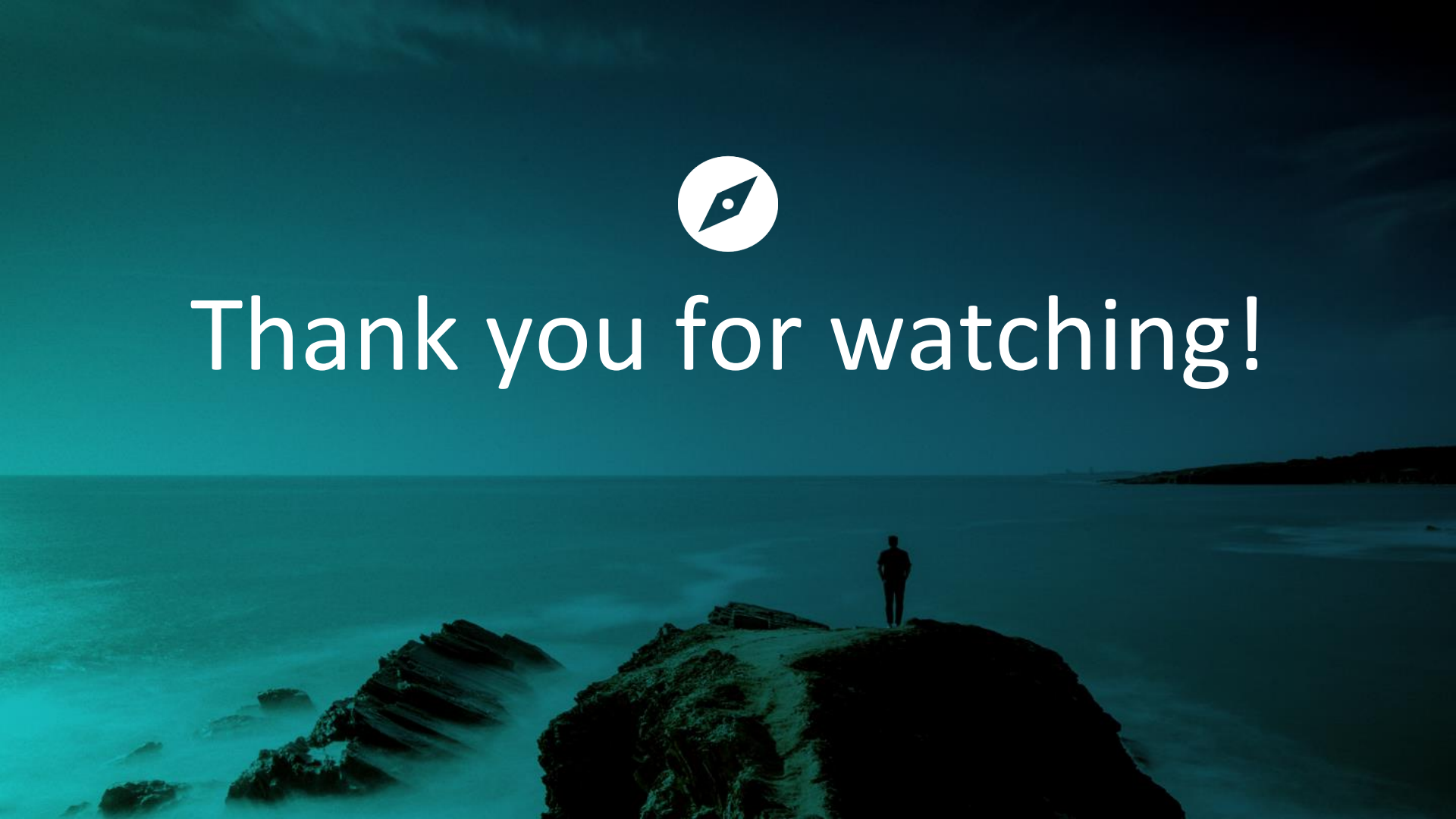
- CloudTrail tracks any CloudFormation API calls in your AWS account.
- Enable logging and specify an Amazon S3 bucket to store the logs which can then be used for auditing purposes.

Code Review and Source Control Your Templates

- This is software after all. Use Software best practices.
- Use code reviews and source control to review changes and to keep an accurate history of your resources.
- With source control you can always revert your stack to a certain version of your template.

Update EC2 Linux Instances Regularly

- Regularly run yum update on your Linux instances to update the RPM package. This insures that you have the latest fixes and security updates.


Thank you for watching!

CloudFormation Deep Dive
CloudFormation CLI

AWS Course Author:
Craig Arcuri

A Bit More on Configuration

- Homebrew:

https://brew.sh/

Homebrew.github.io

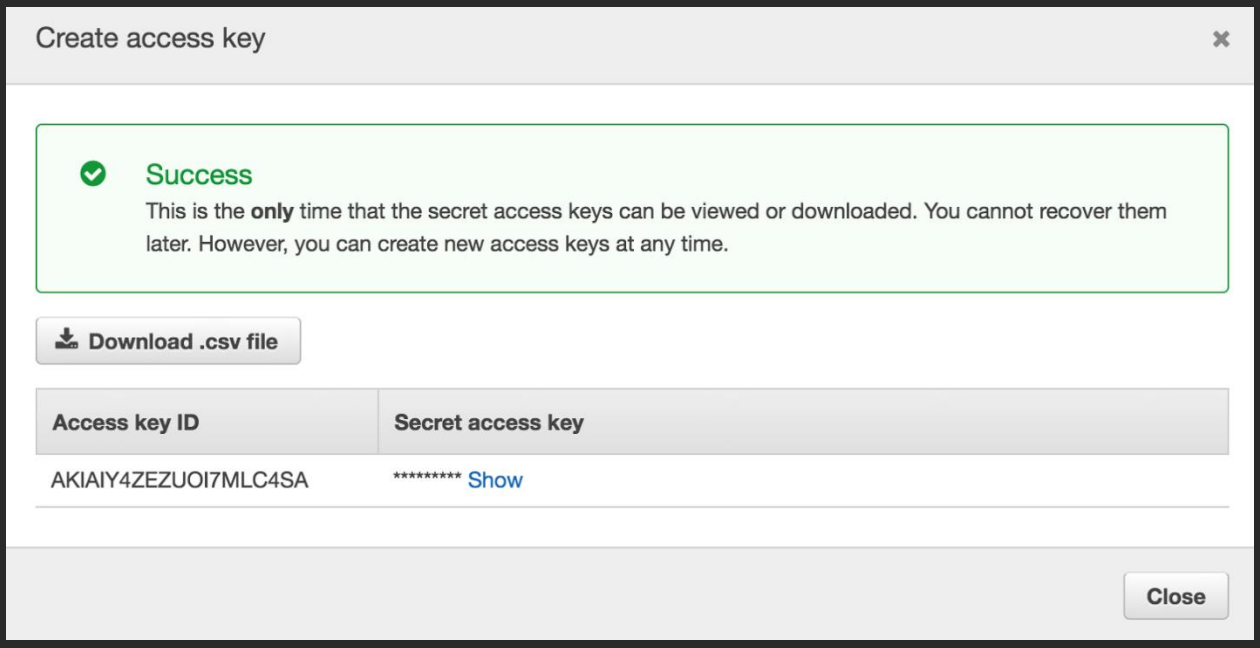
Commands:

brew update

brew install awscli

$ **aws configure**
AWS Access Key ID [None]: ***AKIAIOSFODNN7EXAMPLE***
AWS Secret Access Key [None]:
wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: ***us-west-2***
Default output format [None]: *ENTER*

A Bit More on Configuration



Good To Go? List-Stacks

- Quick Check:
- You can use two AWS CLI commands to get information about your AWS CloudFormation stacks:
- aws cloudformation list-stacks
- aws cloudformation describe-stacks

- **list-stacks** – enables you to get a list of any of the stacks you have created.
- Deleted stacks will be listed up to 90 days from deletion.
- option to filter results by stack status: CREATE_COMPLETE, DELETE_COMPLETE
- Returns name, stack identifier, template, and status
- Examples:

aws cloudformation list-stacks

aws cloudformation list-stacks --stack-status-filter CREATE_COMPLETE

aws cloudformation describe-stacks

- Provides information on your running stacks.
- You can filter results on a stack name
- returns information about the stack, including the name, stack identifier, and status.
- Example:

aws cloudformation describe-stacks --stack-name myteststack

- If you don't use the --stack-name option to limit the output to one stack, information on all your running stacks is returned.

- You can specify one or more stack status codes to list only stacks with the specified status codes.

Creating A Stack

- Command:

aws cloudformation create-stack

- You must provide the stack name, the location of a valid template, and any input parameters.

- Parameters are separated with a space and the key names are case sensitive.

- Example:

```
aws cloudformation create-stack  --stack-name startmyinstance
  --template-body file://home/ec2-user/templates/startmyinstance.json
  --parameters  ParameterKey=KeyPairName,ParameterValue=MyKey
ParameterKey=InstanceType,ParameterValue=t1.micro
```

If you specify a local template file, CloudFormation uploads it to an Amazon S3 bucket in your AWS account.

Viewing Stack History

- You can track the status of the resources AWS CloudFormation is creating and deleting
- aws cloudformation describe-stack-events
- Example:

aws cloudformation describe-stack-events --stack-name S3RetentionSample

- You can run the aws cloudformation describe-stack-events command while the stack is being created to view events as they are reported.

Listing Resources

- Immediately after running the create-stack command, you can list it's resources using:

aws cloudformation list-stack-resources

- lists a summary of each resource in the stack that you specify
- Example:

aws cloudformation list-stack-resources --stack-name S3RetentionSample

Retrieving A Template

- CloudFormation stores the template you use to create your stack as part of the stack.
- You can retrieve the template
- aws cloudformation get-template
- Example:

aws cloudformation get-template --stack-name S3RetentionSample

Output to a file:

aws cloudformation get-template --stack-name S3RetentionSample

Validating A Template

- You can validate your template (check for syntax errors).

aws cloudformation validate-template

- Only checks syntax!
- To check the operational validity, you need to attempt to create the stack.
- You can validate templates locally by using the --template-body parameter
- remotely with the --template-url parameter
- Example:

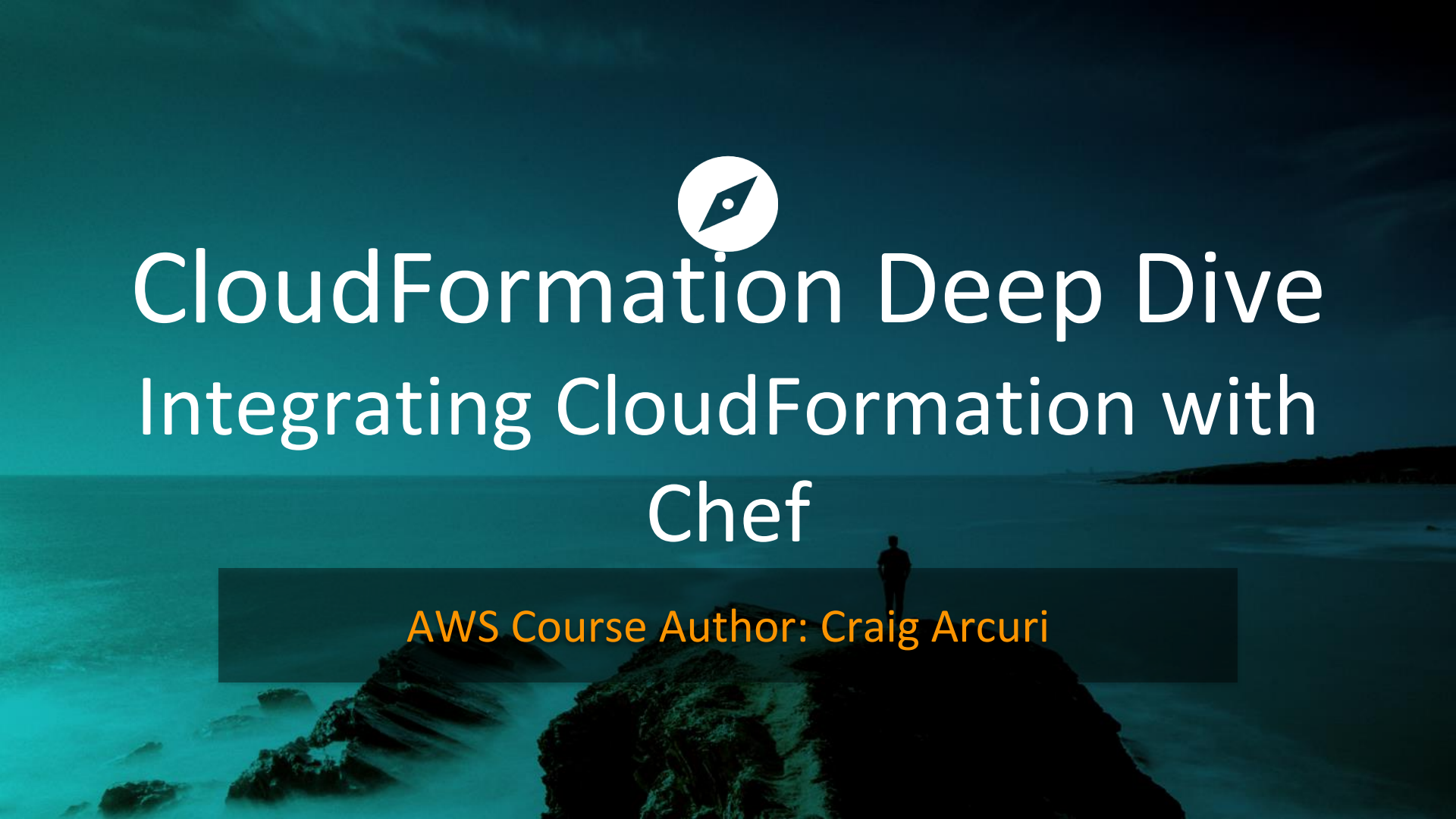
aws cloudformation validate-template --template-url https://s3-us-west-2.amazonaws.com/cloudformation-templates-us-west-2/DynamoDB_Table.template

Deleting a Template

- specify the name of the stack that you want to delete
- you delete the stack and all of its resources.
- Example:

aws cloudformation delete-stack --stack-name myteststack


Thank you for watching!



CloudFormation Deep Dive

Integrating CloudFormation with Chef

AWS Course Author: Craig Arcuri

Chef Server on AWS With CloudFormation

- Chef Server is the highly scalable foundation of the Chef automation platform.
- Use Chef Server to create and manage dynamic infrastructure that runs on the AWS cloud, or manage the servers in your on-premises data center.
- AWS has a Quick Start for Chef Server - Uses AMIs from the AWS Marketplace. You must subscribe to Chef Server from the AWS Marketplace before you launch the Quick Start.

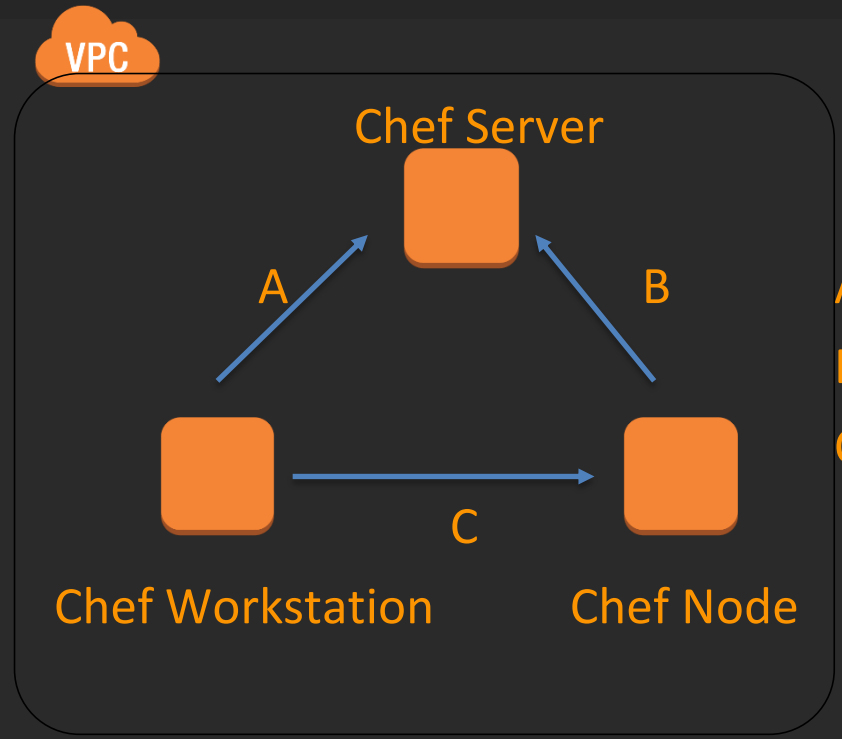
Benefits of Chef on AWS

- Advantages in pricing, automation, resource management.
- Leverage hourly billing, and to pay only for what you use.
- Consolidate your Chef and AWS costs into a single monthly bill.
- Chef can fully support automation in your hybrid deployment.

Chef Integration With CloudFormation

- Automate the deployment of your software applications on Amazon EC2 instances.
- Consistently deploy and configure your AWS resources, along with the software applications that run on top of AWS, all from a single AWS CloudFormation template.

Chef Server Architecture on AWS



A – Knife uploads cookbooks

B – Chef client processes runlist

C – Bootstrap nodes

Quick Start Architecture for Chef Server on AWS

- VPC is created in the region you choose when you launch the stack.
- A single public subnet is created in the first Availability Zone.
- Chef Server is deployed into the VPC subnet with an EIP.
- During instance launch, Chef Server is bootstrapped and the **marketplace-setup** command is run to configure the server (The CloudFormation Template has parameters you need to provide at stack launch.
- A Chef workstation is deployed into the VPC subnet. Git and the Chef Development Kit (Chef DK) are automatically installed on this machine via the AWS CloudFormation bootstrapping process (the workstation is optional, you can instead use a workstation in your own environment).
- An Ubuntu Server node is deployed into the VPC subnet. You can configure a local Git repository (chef-repo) on the workstation, create and upload a cookbook to Chef Server, and then bootstrap the node and run the cookbook to configure a basic web server. But the node is optional also.

Deploying Chef With CloudFormation

- Create a Key Pair in the region.
- Visit the AWS Marketplace and subscribe to the Chef AMI.
- Launch the AWS CloudFormation template into your AWS account to create the Chef stack.
- Note: The stack takes approximately 35 minutes to create.

Application Deployment With Chef

- You can automate the deployment of your software applications on EC2 instances instead of manually building various scripts.
- Combine Chef with CloudFormation to consistently deploy and configure your resources and the software applications that run on top of it.
- Example: Deploy a Wordpress Blog with backend database. Use an AWS CloudFormation template to provision a web server and database.
- Also pass in metadata and commands to your web server to install Chef.
- With a Chef recipe, you can install and configure the Wordpress blog on your web server.
- You can reuse the template and maintain a configuration history.
- You can bootstrap your instances or pre-bake your AMIs.

Application Deployment Options

- Build custom AMIs.
- Run user data scripts.
- Config management tools – Chef, Puppet

CloudFormation and Chef – Wordpress Blog

- Fault-tolerant, load-balancing web server that runs a WordPress blog with a back-end database.
- Web server is an Amazon EC2 instance in an Auto Scaling group.
- Auto Scaling group sits behind a load balancer, which gives the WordPress blog a single endpoint.
- After the Amazon EC2 instance is started, AWS CloudFormation installs and configures Chef on the instance.
- It then runs Chef to install and configure WordPress.
- Chef local mode uses a local Chef repository so that you can use recipes and Chef without a Chef server.

CloudFormation and Chef –

- The sample template specifies a desired group size of one instance.
- Auto Scaling can automatically stop unhealthy instances and launch new instances, ensuring that the WordPress blog isn't down for long periods of time.
- ASG in the template:

```
"WebServerGroup" : {
   "Type" : "AWS::AutoScaling::AutoScalingGroup",
    "Properties" : {
     "AvailabilityZones" : { "Fn::GetAZs" : "" },
     "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },
     "MinSize" : "1",
     "MaxSize" : "5",
     "DesiredCapacity" : { "Ref" : "WebServerCapacity" },
     "HealthCheckType" : "ELB",
     "LoadBalancerNames" : [ { "Ref" : "ElasticLoadBalancer" } ]
   }
 }
```

- HealthCheckType - Specifies whether to use just the Auto Scaling health check (the Amazon EC2 instance status checks) or both the Auto Scaling and load balancer health check.

The Mechanics of CloudFormation and Chef

- The sample template specifies a desired group size of one instance.
- Auto Scaling can automatically stop unhealthy instances and launch new instances, ensuring that the WordPress blog isn't down for long periods of time.
- ASG in the template:

```
"WebServerGroup" : {
   "Type" : "AWS::AutoScaling::AutoScalingGroup",
    "Properties" : {
     "AvailabilityZones" : { "Fn::GetAZs" : "" },
     "LaunchConfigurationName" : { "Ref" : "LaunchConfig" },
     "MinSize" : "1",
     "MaxSize" : "5",
     "DesiredCapacity" : { "Ref" : "WebServerCapacity" },
     "HealthCheckType" : "ELB",
     "LoadBalancerNames" : [ { "Ref" : "ElasticLoadBalancer" } ]
   }
 }
```

- HealthCheckType - Specifies whether to use just the Auto Scaling health check (the Amazon EC2 instance status checks) or both the Auto Scaling and load balancer health check.

The Mechanics of CloudFormation and Chef - 2

- A launch configuration describes the EC2 instances that are to be launched in the Auto Scaling group. Specify properties such as the instance type, the AMI ID, and bootstrapping information.
- User Data:

```
"UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
                "#!/bin/bash –xe\n",
                 "yum update aws-cfn-bootstrap\n",
                 "/opt/aws/bin/cfn-init ",
                 "        --stack ", { "Ref" : "AWS::StackName" },
                "        --resource LaunchConfig ",
                "        --configsets wordpress_install ",
                "        --region ", { "Ref" : "AWS::Region" }, "\n",
                 "/opt/aws/bin/cfn-signal –e $? '", { "Ref" : "WebServerWaitHandle" }, "'\n"
```

- Sending a shell script to the instance. The script updates the AWS CloudFormation bootstrapping tools and then calls cfn-init.

Launch Configuration and CloudFormation::Init

- The launch configuration includes a metadata section with the AWS::CloudFormation::Init resource.
- AWS::CloudFormation::Init resource defines a set of tasks to perform when used in combination with the cfn-init helper script:

Download and install Chef

Download a WordPress cookbook

Install WordPress

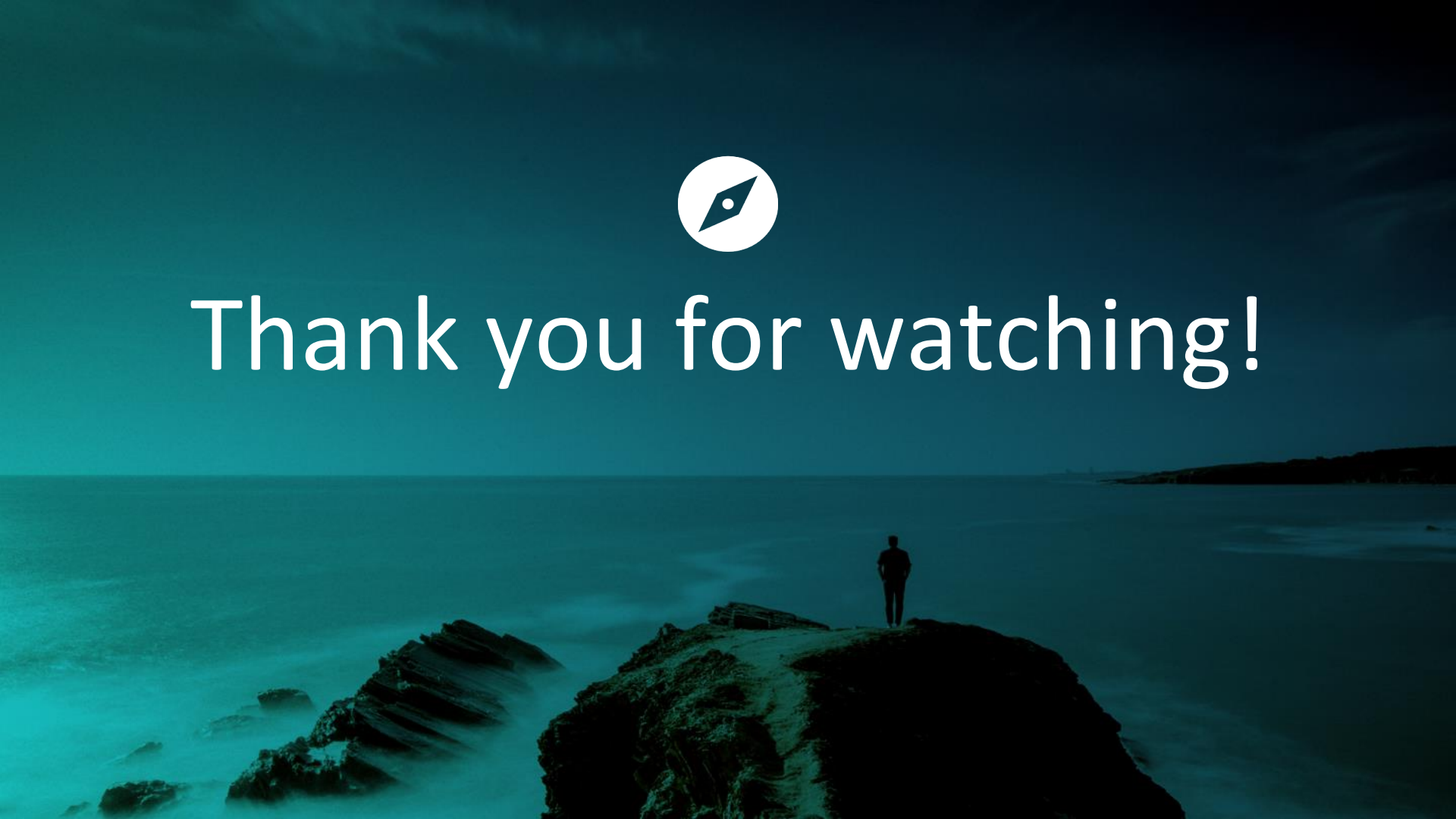
- In AWS::CloudFormation::Init resource, a group of tasks are contained within a configuration. To determine which configurations to use and in what order to run them, you list configurations in a configuration set:

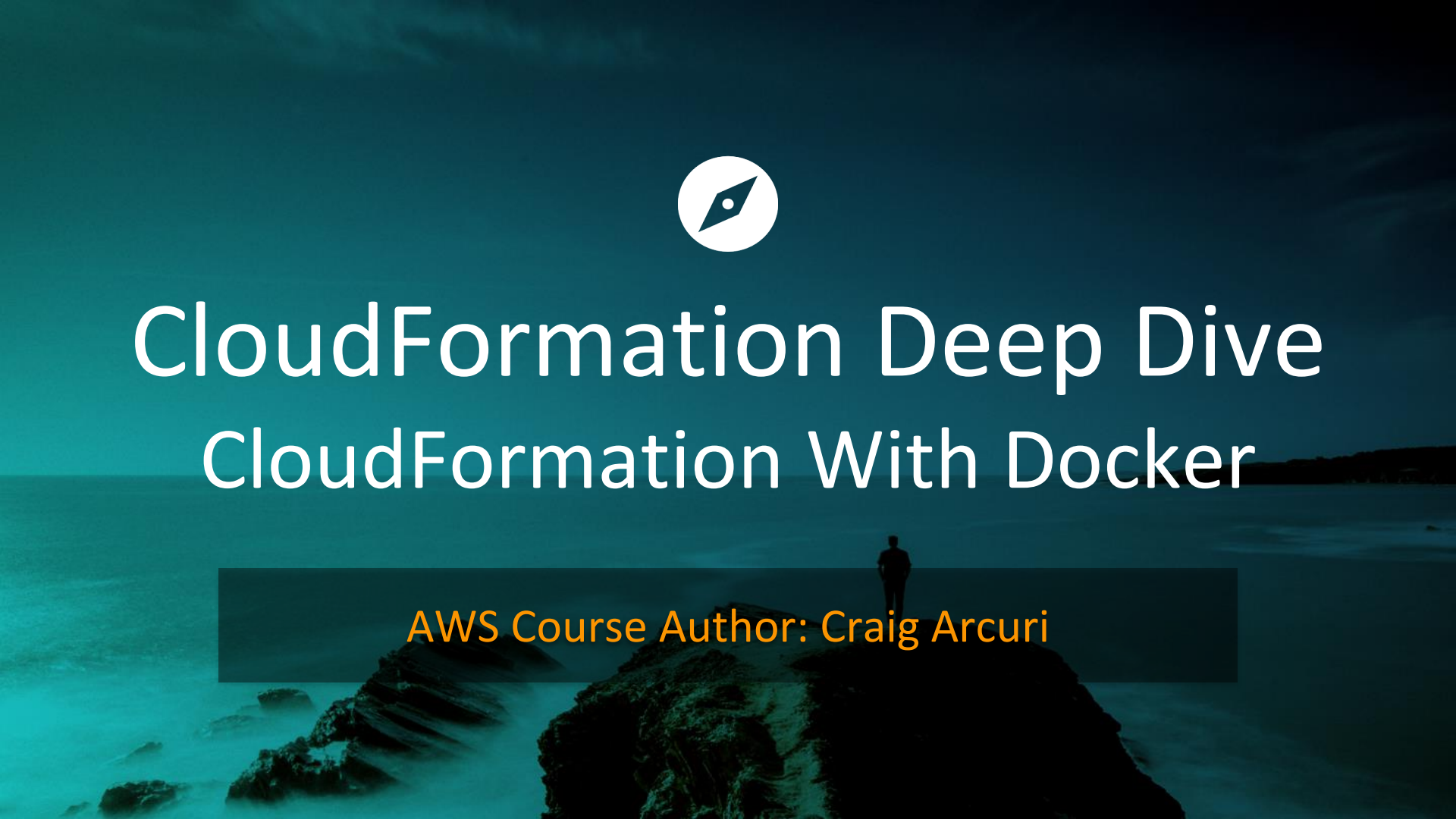
"configSets" : { "wordpress_install" : ["install_cfn", "install_chefdk", "install_chef", "install_wordpress", "run_chef"] },

- chefdk downloads and installs the chef development kit.

Launch Configuration and CloudFormation::Init - 2

- The install_chef configuration sets up a local Chef repository on the instance, writes files on the instance: a Chef installation file (install.sh), a Knife configuration file (knife.rb), and a Chef client configuration file (client.rb).
- install_wordpress configuration installs WordPress by using a WordPress cookbook.
- The run_chef configuration runs the Chef client in local mode to install and configure WordPress.
- Not only can AWS CloudFormation set up and configure your AWS resources, you can also integrate it with other tools, such as Chef, to help bootstrap your instances. This integration can help you reliably and consistently deploy any web application on your Amazon EC2 instances.


Thank you for watching!



CloudFormation Deep Dive

CloudFormation With Docker

AWS Course Author: Craig Arcuri

What Is Docker?

- The leading software containerization platform
- A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings.
- Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.
- Docker containers running on a single machine share that machine's operating system kernel; they start instantly and use less compute and RAM.
- Containers and Virtual Machines have similar resource isolation and allocation benefits. But containers virtualize the operating system instead of hardware.
- Allows you to build, test, and deploy applications quickly.

Docker On AWS

- Docker on AWS provides developers and admins a highly reliable, low-cost way to build, ship, and run distributed applications at any scale.
- AWS supports both Docker licensing models: open source Docker Community Edition (CE) and subscription-based Docker Enterprise Edition (EE).
- Docker for AWS uses CloudFormation to provision a set of autoscaling EC2 instances that run a Docker Swarm cluster.
- Docker on AWS setup also provisions other AWS-related resources related to the cluster: networking, load balancing, message-queuing services, access management controls, and CloudWatch logging.
- AWS does AWS this like autoscaling of the cluster, Docker does Docker things like deployment of apps.

Docker Setup

- Docker Enterprise Edition (EE) for AWS - fully baked and tested, and comes with the latest Docker Enterprise Edition for AWS.
- Docker Community Edition (CE) for AWS – can use Stable Channel (fully baked and tested, and comes with the latest CE version of Docker) or Edge Channel (offers cutting edge features of the CE version of Docker and comes with experimental features turned on).
- Two ways to deploy Docker for AWS:
  - With a pre-existing VPC
  - With a new VPC created by Docker
- It is recommended to allow Docker for AWS to create the VPC since it allows Docker to optimize the environment.

Docker Setup - Continued

- Create New VPC - Creates a new VPC, subnets, gateways, and everything else needed in order to run Docker for AWS. It is the easiest way to get started, and requires the least amount of work. A CloudFormation Template is provided.
- Install with an Existing VPC – Create a VPC in your selected region and set it up with Internet Gateway, Subnets, and Route Tables. Need to have three different subnets, ideally each in their own availability zone. Launch the docker for AWS CloudFormation stack, make sure to use the one for existing VPCs.

Docker Prerequisites

- Access to an AWS account with permissions to use CloudFormation and creating the following objects:
- EC2 instances + Auto Scaling groups
- IAM profiles
- DynamoDB Tables
- SQS Queue
- VPC + subnets and security groups
- ELB
- CloudWatch Log Group

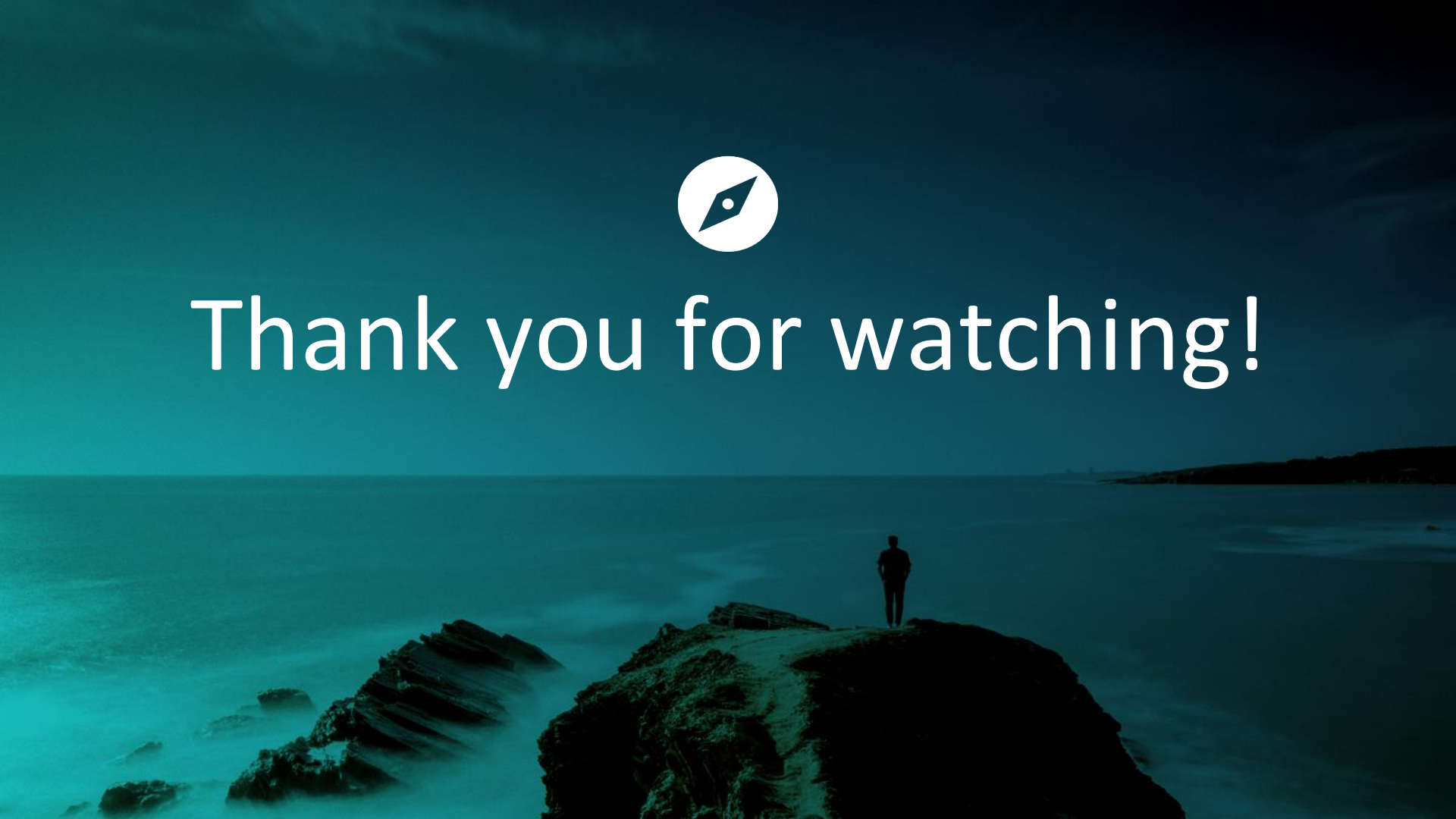
- Need an SSH Key in the region.

Docker Configuration

- Installed with a CloudFormation template that configures Docker in swarm mode, running on instances backed by custom AMIs.
- A Swarm is a cluster of Docker engines that can be managed as a single system.
- Two ways to deploy Docker for AWS:
    - AWS Management Console
    - AWS CLI

Docker Configuration Options

- **KeyName -** Pick the SSH key that will be used when you SSH into the manager nodes.
- **InstanceType –** Instance type for worker nodes.
- **ManagerInstanceType -** The EC2 instance type for your manager nodes. The larger your swarm, the larger the instance size you should use.
- **ClusterSize -** The number of workers you want in your swarm (0-1000).
- **ManagerSize -** The number of managers in your swarm (1,3, or 5 managers).
- **EnableSystemPrune -** Enable if you want Docker to automatically cleanup unused space on your swarm nodes.
- **EnableCloudWatchLogs - S**ends your container logs to CloudWatch if enabled.


Thank you for watching!

CloudFormation Deep Dive

CloudFormation With Puppet

AWS Course Author: Craig Arcuri

What is Puppet?

- Puppet is an open source platform for provisioning, configuring and patching applications and operating system components.
- 2 basic building blocks:
- Puppet Master: Centralized configuration server that holds the definitions and instructions needed to install applications as well as server roles.
- Puppet Client: Connects to a Puppet server to download the necessary instructions to install, update and patch the software running on it.

CloudFormation Bootstrapping and Puppet

- CloudFormation provisions your resources, it raises the obvious question of how is your application software deployed, configured and executed on the EC2 instances?
- CloudFormation bootstrap helper scripts to deploy applications using Puppet.

Puppet Master

- Central location for managing you application and OS component configuration
- With a CloudFormation Template you can bootstrap a Puppet Master on an AMI.
- Template would take to parameters that are used to populate the Puppet master with a set of modules or application and OS component configurations that can be downloaded to Puppet clients:
- ContentLocation - points to an archive (.zip or tar'd and zipped) file that contains the manifests, templates and other artifacts making up the modules.
- ContentManifest - contains the mapping between server roles and the modules needed.
- The template can be run in any region, with any instance type being used to host the Puppet Master. Mappings are used to define the architecture of the instance and to select the correct EC2 AMI based on architecture and region.

Puppet Master

- The template defines an IAM user that only has permissions to call the CloudFormation DescribeStackResource API. This is passed to the Cloud-init script for the Puppet Master EC2 instance.
- The Cloud-init script calls cfn-init to install the Puppet Master and populate it with an initial set of modules and then signals completion when the Puppet Master is ready to accept requests from Puppet clients.
- A Facter plug-in is also deployed to the Puppet Master at /etc/puppet/modules/cfn. Allows the client template to specify the server role and any properties needed by the modules in template metadata.

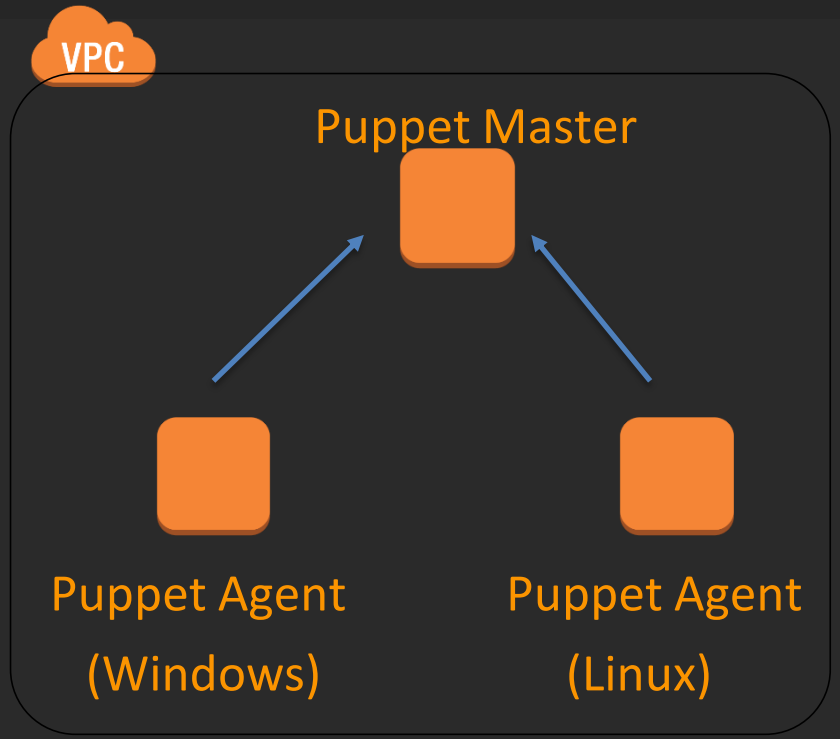
Puppet Client

- Puppet Client Template creates an Amazon EC2 instance running the Puppet client that can be configured to install a server role from the Puppet Master.
- Could also create a highly available, multi-AZ application by using an Auto Scaling group rather than a single Amazon EC2 instance.
- The template takes 2 input parameters PuppetClientSecurityGroup and PuppetMasterDNSName to identify and access the Puppet Master.
- The template also takes 2 input parameters StackNameOrId and ResourceName that are used to reference the server roles defined in resource metadata in the calling template.
- Cloud-init script calls cfn-init twice, the first time using the metadata defined in the template to install the Puppet client and a second time to configure the roles defined in the calling template.
- A WaitCondition is used to delay the stack being created until the Puppet client is configured and running.

Puppet Quickstart on AWS

- Deploy and test a Puppet master and Puppet agents on the Amazon Web Services (AWS) cloud.
- Deployment takes approximately 20 minutes.
- Every step of your software delivery process, from provisioning instances to orchestration and reporting, including production release of software and updates, can be automated.
- Puppet uses a client/server model where agent nodes get configuration profiles from the Puppet master, which is a server that controls the configuration information.
- Puppet enables you to define configurations that are idempotent, meaning they can be run multiple times without any risks.
- Once you've developed your configurations, your agents can apply the configuration on a regular interval (30 minutes by default), which will keep your systems in their desired state. If your system state drifts out of the desired configuration, the Puppet agent will re-apply your configuration.

Puppet Architecture on AWS



Puppet Deployment on AWS

- A VPC is created in the region you choose when you launch the stack.
- A single, public VPC subnet is created in the first Availability Zone.
- One Puppet master is deployed into the VPC subnet.
- During instance launch, the Puppet master is bootstrapped to automatically install all required software along with Puppet modules and manifests that can be used to configure the Puppet agents.
- One Ubuntu Server is deployed into the VPC subnet.
- You apply a web server configuration that will install and configure the Apache web server and PHP.
- One Windows Server 2012 R2 server is deployed into the VPC subnet.
- You can apply a web server configuration that will install and configure the Internet Information Services (IIS) web server and ASP.NET.

Puppet Master Installation

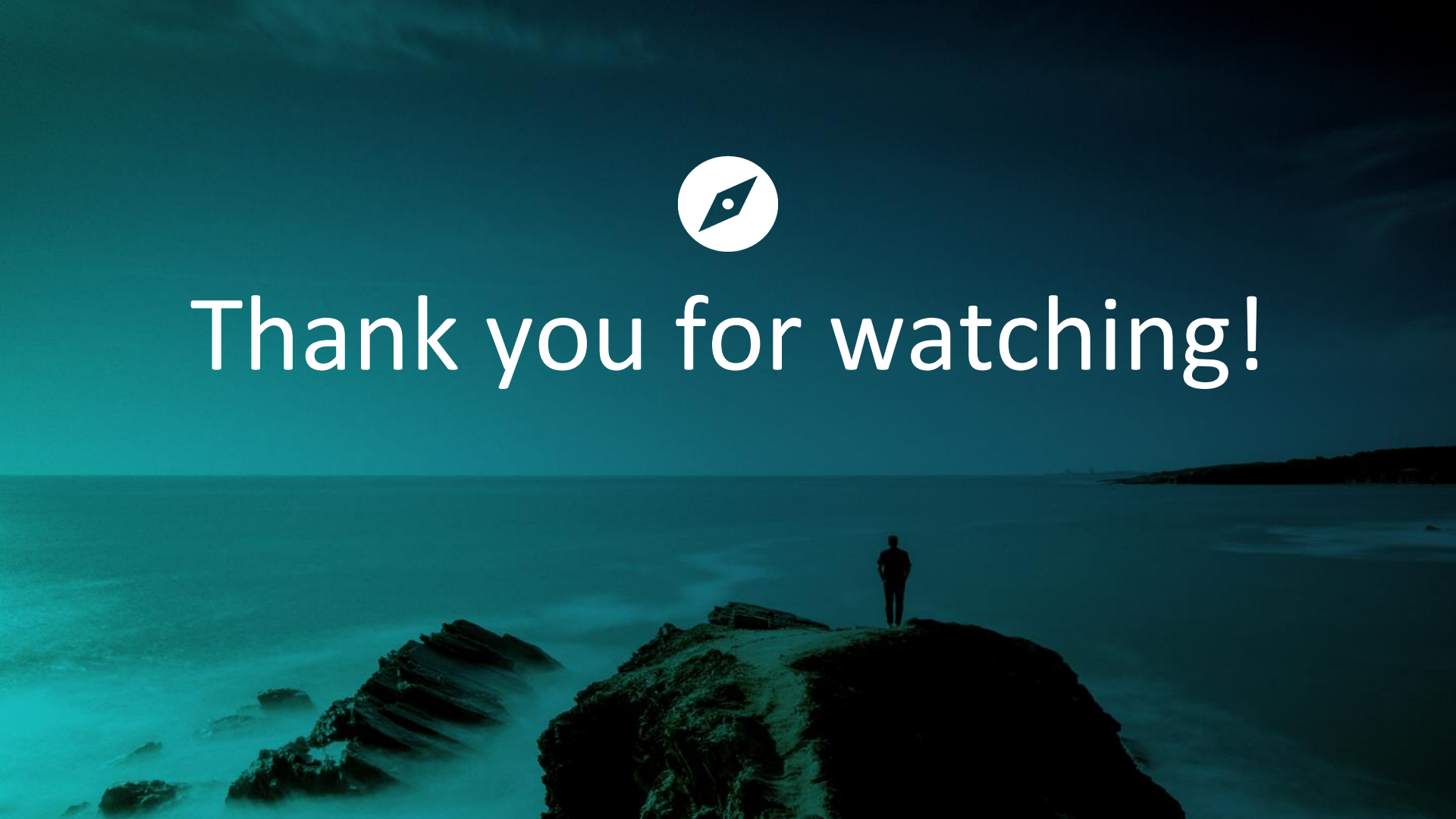
- Deploys the Puppet master on an EC2 instance that is running Ubuntu.
- The installation is automated with a user data script that executes when the instance is launched via AWS CloudFormation.
- The Open Source version of Puppet is installed using a package called *puppetmaster-passenger.*
- Preconfigured Puppet modules from S3, which will enable you to apply a web server configuration to both the Windows and Linux nodes.
- The Puppet master acts as a certificate authority (CA), and SSL certificates are used to authenticate communications between the master and agent nodes.
- Quick Start pre-provisions record sets for each EC2 instance in Route 53, the Puppet master will use the host name *puppet.example.com* by default.
- Puppet agents need to be configured to connect to your Puppet master, and the Quick Start automates that work.
- The first time the Puppet agent runs on a node, it will send a certificate signing request to the master.

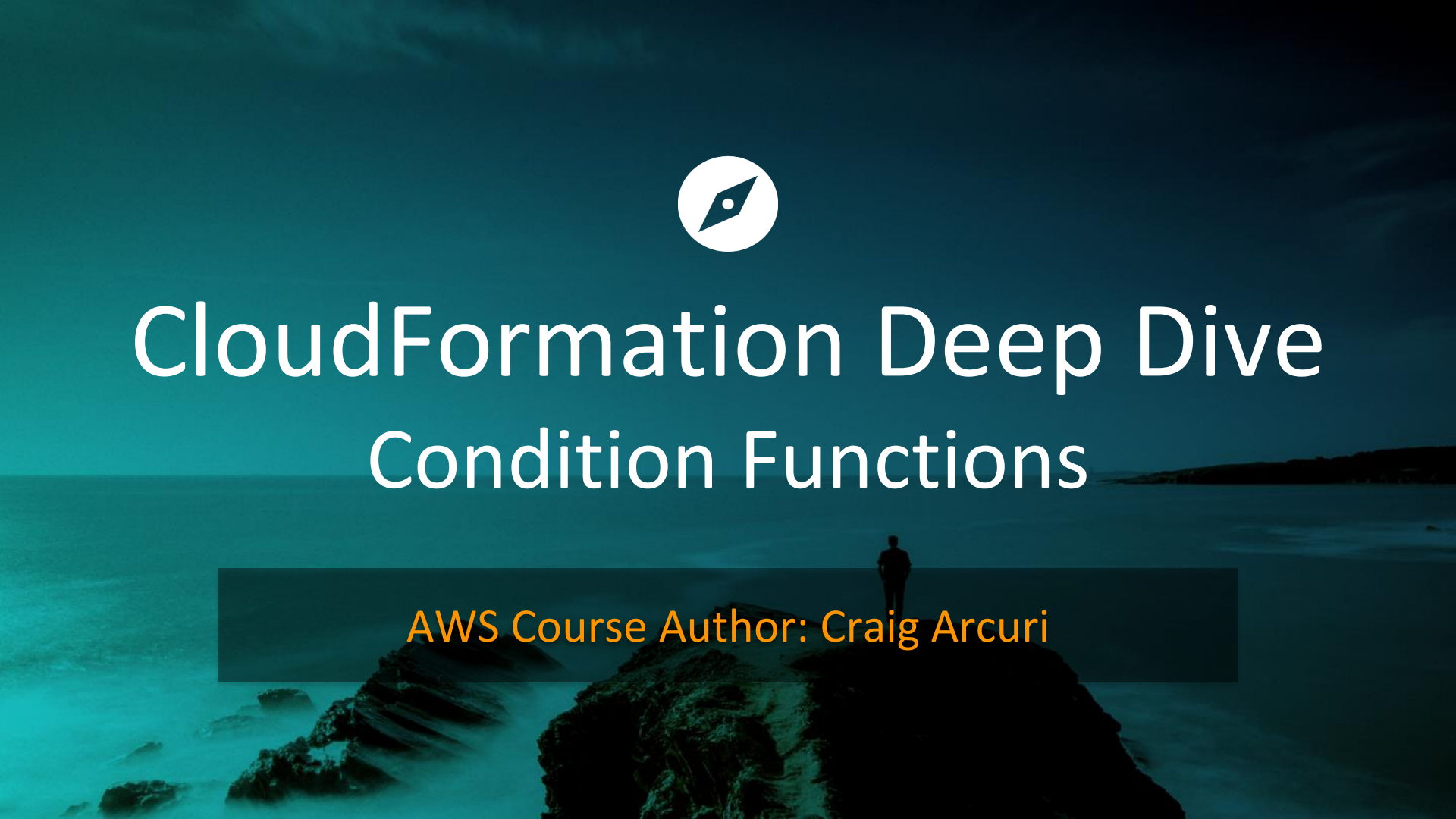
Puppet Agent Installation

- Linux agent deployed by this Quick Start also runs Ubuntu.
- The Quick Start runs a user data script when it launches the agent via CloudFormation.
- This script installs the agent and configures it to point to the master at *puppet.example.com*, and the server automatically requests and signs the agent certificate.
- The Windows agent is deployed on an instance running Windows Server 2012 R2. A user data script is run to install and configure the agent at launch, after the master has already been deployed. The Quick Start automatically downloads and installs the *puppetlabs-powershell* and *puppetlabs-windowsfeature* modules from the Puppet Forge. These modules are used within a module manifest that installs the IIS web server with all required components and support for ASP.NET websites.

Puppet Deployment And Configuration

- **Prerequisites** - Set up and enable name resolution via DNS. This enables communication between the Puppet Master and Puppet Agent.
    - The Puppet master must be reachable by agents via TCP port 8140.
    - Make sure you can use Secure Shell (SSH) or Remote Desktop Protocol (RDP) for remote connections.
- Launch the AWS CloudFormation template into your AWS account, specify parameter values, and create the stack.
- Configure Puppet Agents - Review the module manifests for the Linux and Windows agents, connect to the agents via SSH or RDP, and apply the configurations.


Thank you for watching!



CloudFormation Deep Dive

Condition Functions

AWS Course Author: Craig Arcuri

Condition Functions

- You can use Condition Functions to conditionally create stack resources.
- Evaluated based on input parameters that you declare when you create or update a stack.
- Associate them with resources or resource properties in the Resources and Outputs sections of a template.
- Define conditions in the Conditions section of the template (with the exception of Fn::If).
- You can use Fn::If condition in the metadata attribute, update policy attribute, and property values in the Resources section and Outputs sections of a template.
- Use conditions when you want to reuse a template that can create resources in different contexts, such as a test environment versus a production environment.

Common Use Case – Prod or Dev?

- Evaluate EnvironmentType input parameter for either prod or dev. Based on the condition input, tailor your deployment for either environment. Prod will probably have more compute power. Save money in Dev.
- Using a condition, you can automate your deployments across multiple environments and cut costs.
- You can only reference other conditions from the Parameters and Mappings sections.
- You cannot reference the logical ID of a resource in a condition.

Associating a Condition

- If you need to conditionally create resources, resource properties, or outputs, you must associate a condition with them.

```
NewVolume:
  Type: "AWS::EC2::Volume"
  Condition: CreateProdResources
  Properties:
    Size: 100
    AvailabilityZone: !GetAtt EC2Instance.AvailabilityZone
```

Fn::If

- You only need to specify the condition name.

Example:

```
NewVolume:
  Type: "AWS::EC2::Volume"
  Properties:
    Size:
      !If [BigGulp, 50, 5]
    AvailabilityZone: !GetAtt: Ec2Instance.AvailabilityZone
  DeletionPolicy: Snapshot
```

Nesting Conditions

- You can use conditions inside of conditions:

```
ParentCondition: !And
  - !Equals ["sg-mysggroup", !Ref "ASecurityGroup"]
  - !Condition ChildCondition
```

Fn::And

- Returns true if all the conditions are true, otherwise returns false.
- The minimum number of conditions that you can include is 2, and the maximum is 10.
- JSON:
- "Fn::And": [{condition}, {...}]
- YAML:

  Fn::And: [condition]
- Or Short form:

  !And [condition]

Fn::Equals

- Compares if two values are equal.
- Returns true or false
- JSON:

  "Fn::Equals" : ["value_1", "value_2"]
- YAML:

  Fn::Equals: [value_1, value_2]

  or

  !Equals [value_1, value_2]

Example:

```
UseProdCondition:
  !Equals [!Ref EnvironmentType, prod]
```

Do or Do not, there is no → !

- The exclamation point indicates the YAML short form of a command. It does NOT indicate a Boolean not!!!!

Fn::If

- Returns one value if the specified condition evaluates to true, and another value if it is false.
- JSON:
- "Fn::If": [condition_name, value_if_true, value_if_false]
- YAML:
- Fn::If: [condition_name, value_if_true, value_if_false]

- Example (YAML short form):
- SecurityGroups:
- - !If [CreateNewSecurityGroup, !Ref NewSecurityGroup, !Ref ExistingSecurityGroup]

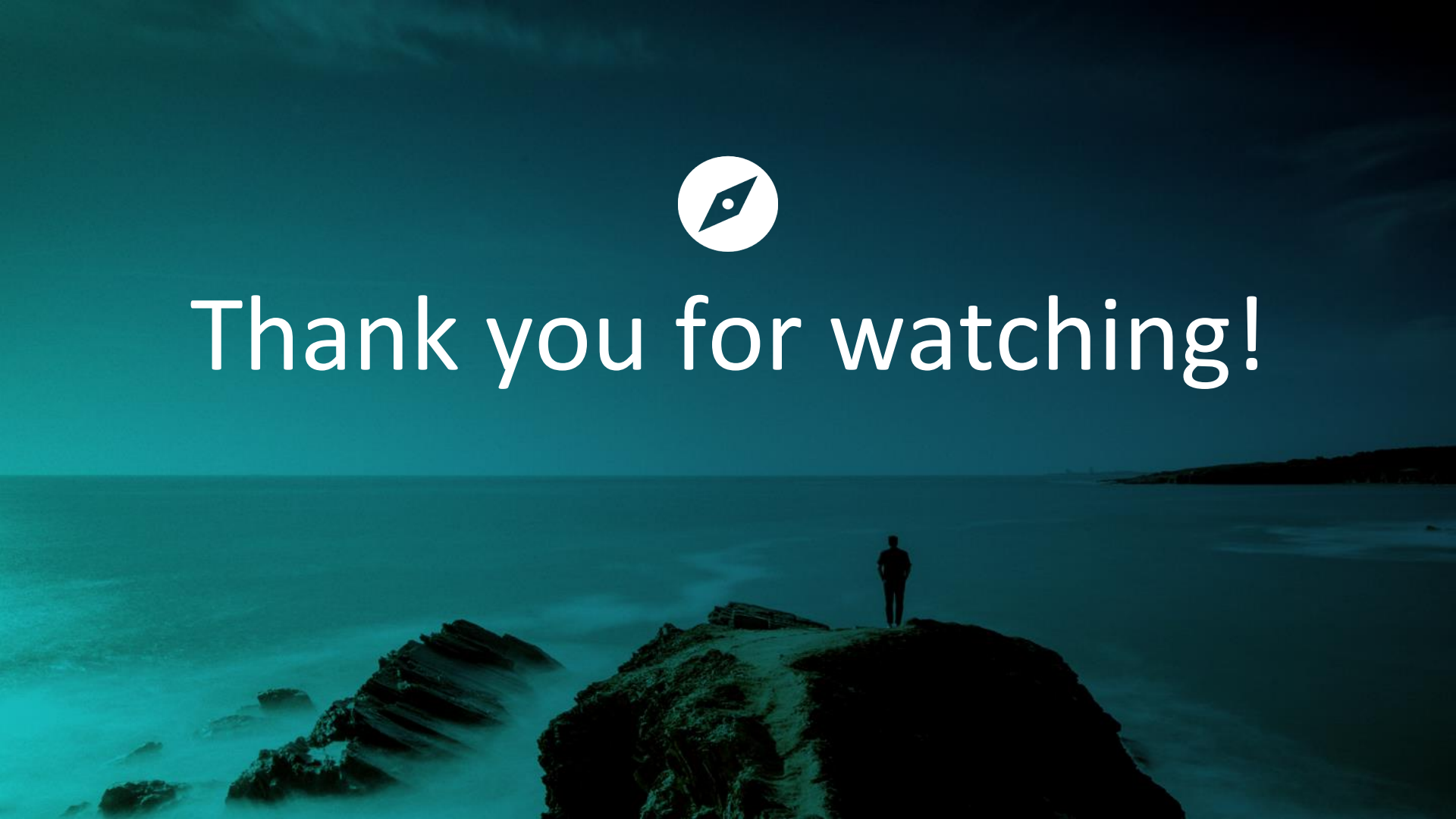
Fn::Not

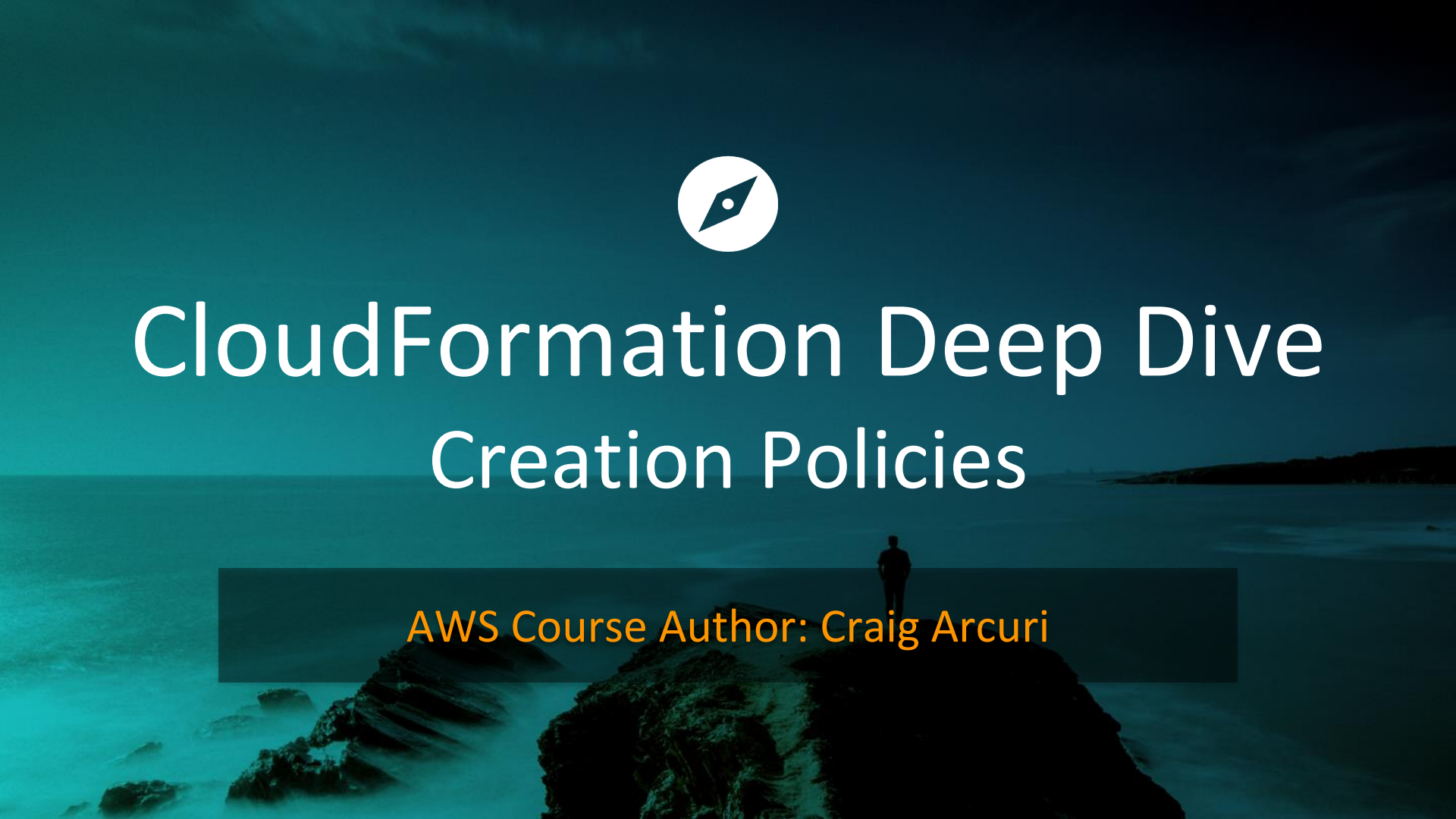
- Returns true if the condition evaluates to false, and returns false if the condition evaluates to true.
- JSON: "Fn::Not": [{condition}]
- YAML: Fn::Not: [condition]
- Short form: !Not [condition]

Fn::Or

- Returns true if any of the conditions are true or returns false if none of the conditions are true.
- YAML:

- ExampleOrCondition:
- !Or [!Equals [sg-mysggroup, !Ref ASecurityGroup], Condition: SomeOtherCondition]


Thank you for watching!



CloudFormation Deep Dive

Creation Policies

AWS Course Author: Craig Arcuri

Creation Policies Defined

- Pause the creation of a resource until a requisite number of success signals are received (sounds like WaitCondition right?)
- For EC2 instances and Auto Scaling groups containing EC2 instance, AWS recommends using a Creation Policy rather than a WaitCondition.
- Example: You are creating EC2 instances, but the instances also require that specific applications (such as Nginx) also be installed. Without a creation policy, CloudFormation would declare the EC2 instance as created before the software installation is completed.
- Creation Policies can guarantee that your application installation or bootstrapping has completed successfully during stack creation.
- Use a Creation Policy – specify the number of success signals to receive before declaring creation complete.
- Just as with a WaitCondition, you also specify the timeout.

Creating A Creation Policy

- You first need to attach the creation policy to a resource in the stack. Understand, the you are changing the behavior of this resource. It will not transition to CREATE_COMPLETE until or if it receives the required number of success signals.
- Success signals can be sent back to the stack through the SignalResource API call or CLI, or with helper scripts.
- Key point: these signals can be viewed in the Events tab during stack creation.

Creation Policy Syntax

- The creation policy only takes effect when CloudFormation creates the instance.
- Syntax (within an Auto Scaling Group):

```
"CreationPolicy" : {
  "AutoScalingCreationPolicy" : {
    "MinSuccessfulInstancesPercent" : Integer
  },
  "ResourceSignal" : {
    "Count" : Integer,
    "Timeout" : String
  }
}
```

Creation Policy Syntax

- A creation policy must be attached to a resource.
- For an Auto Scaling group, the creation policy will typically be waiting on success signals from the creation of EC2 instances up to the Desired Capacity of the ASG. This must take place before the timeout period for CREATE_COMPLETE
- AutoScalingCreationPolicy – only needed for ASGs
- MinimumSuccessfulInstancesPercent – specifies a percentage of instances
- Count – number of signals required
- Time period in which the requisite number of success signals must be received.

Creation Policy Added To A Resource

```
"AutoScalingGroup": {
      "Type": "AWS::AutoScaling::AutoScalingGroup",
      "Properties": {
       ... },
 "CreationPolicy": {
      "ResourceSignal": {
      "Count": "3",
      "Timeout": "PT5M“
      }
   }
 }
```

Signaling A Resource

```
"UserData": {
      "Fn::Base64": {
            "Fn::Join" : [ "", [
                  "#!/bin/bash -xe\n",
                  "yum install -y aws-cfn-bootstrap\n",
                  "/opt/aws/bin/cfn-signal -e 0 --stack ", { "Ref": "AWS::StackName" },
                  " --resource AutoScalingGroup ",
                  " --region ", { "Ref" : "AWS::Region" }, "\n“
      ] ]
   }
}
```

Signaling A Resource

- "/opt/aws/bin/cfn-signal -e 0 --stack ", { "Ref": "AWS::StackName" }, - This is a helper script that sends a success signal back to the stack after running bash commands.
- --stack ", { "Ref": "AWS::StackName" }, - This command IDs the stack we want to signal.
- --resource AutoScalingGroup – The logical ID of the resource
- CloudFormation init and signal commands (cfn-init and cfn-signal) can be used to send a signal back to the resource.
- cfn-init can be used to install the app on an EC2 instance.
- Metadata can contain instructions on what needs installed/created.

Cfn-init Config Sets

```
“/opt/aws/bin/cfn-init “,
“       --stack “, { “Ref” : “AWS::StackName” },
“       --resource MyInstance “,
“       --configsets InstallRun “,
“       --region “, { “Ref” : “AWS::Region” }, “\n”
```

- With configSets you can group metadata config keys to specify an order of configuration.

1 download and install applications 2 configure them 3 start services

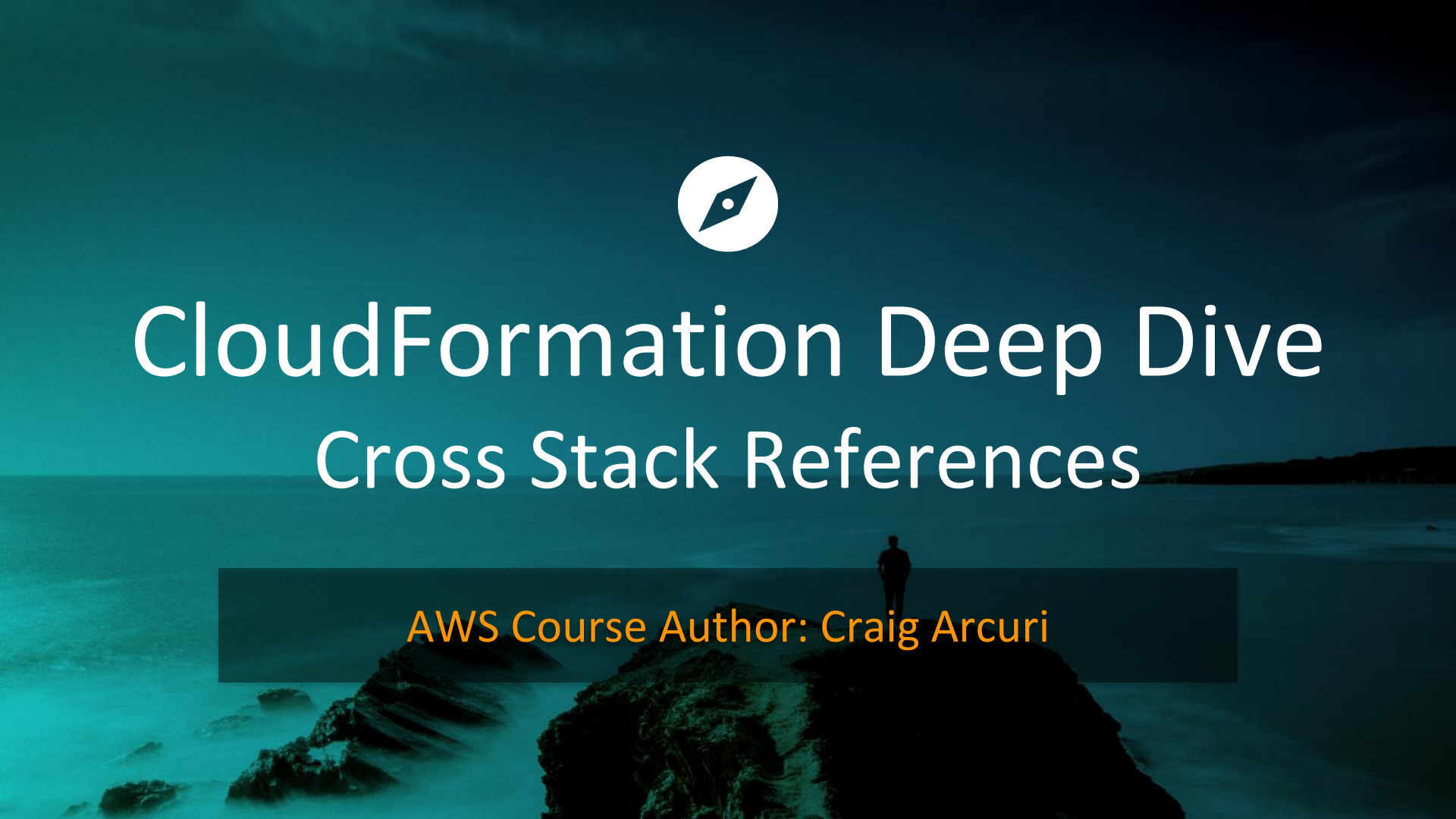
Metadata Config Sets

```
"Metadata" : {
    "AWS::CloudFormation::Init" : {
        "configSets" : {
            "InstallAndRun" :  [ "Install", "Configure", "Run"]
        },
        "Install": {....},
        "Configure": {....},
        "Run": {....}
    }
}
```

- Will perform in the order specified by the config keys (Install, Config, Run)

Creation Policies and Wait Conditions

- Can use together to track the progress of bootstrapping an instance.
- Coordinate the creation of different resources:
    - After an action on an instance completes, trigger the creation of another dependent resource.



CloudFormation Deep Dive

Cross Stack References

AWS Course Author: Craig Arcuri

Exporting Stacks

- What are Cross-Stack references? A CloudFormation feature which allows you to import and export stacks values between stacks.
- Why use them? Well, this is code after all. What are some of the key paradigms of software? Modularization, code reuse.
- Template can become very long and unmanageable. Break them up! Modularize. Share data between stacks.
- Split up work amongst team members. Network teams and DB teams can work on separate templates but share data. Database team needs to know the subnet that will house their DB servers.
- Export this data from the network stack!

Cross-Stack References - Export

- Cross-Stack references are imports and exports between stacks.
- Very simple and easy to use concept but very powerful.
- Example:

```
Outputs:
  SubnetID1:
      Value: !Ref Subnet1
      Export: Name: ExampleSubnet
```

- The export is implemented with just one line in the output section.

Cross-Stack References - Import

- Note: the Export name must be unique within the AWS account and region.
- Now you can import the value into your other stacks:

```
EC2Instance:
    Type: AWS::EC2::Instance
        Properties:
            SubnetID:
                - !ImportValue ExampleSubnet      #note:    (Fn::ImportValue)
```

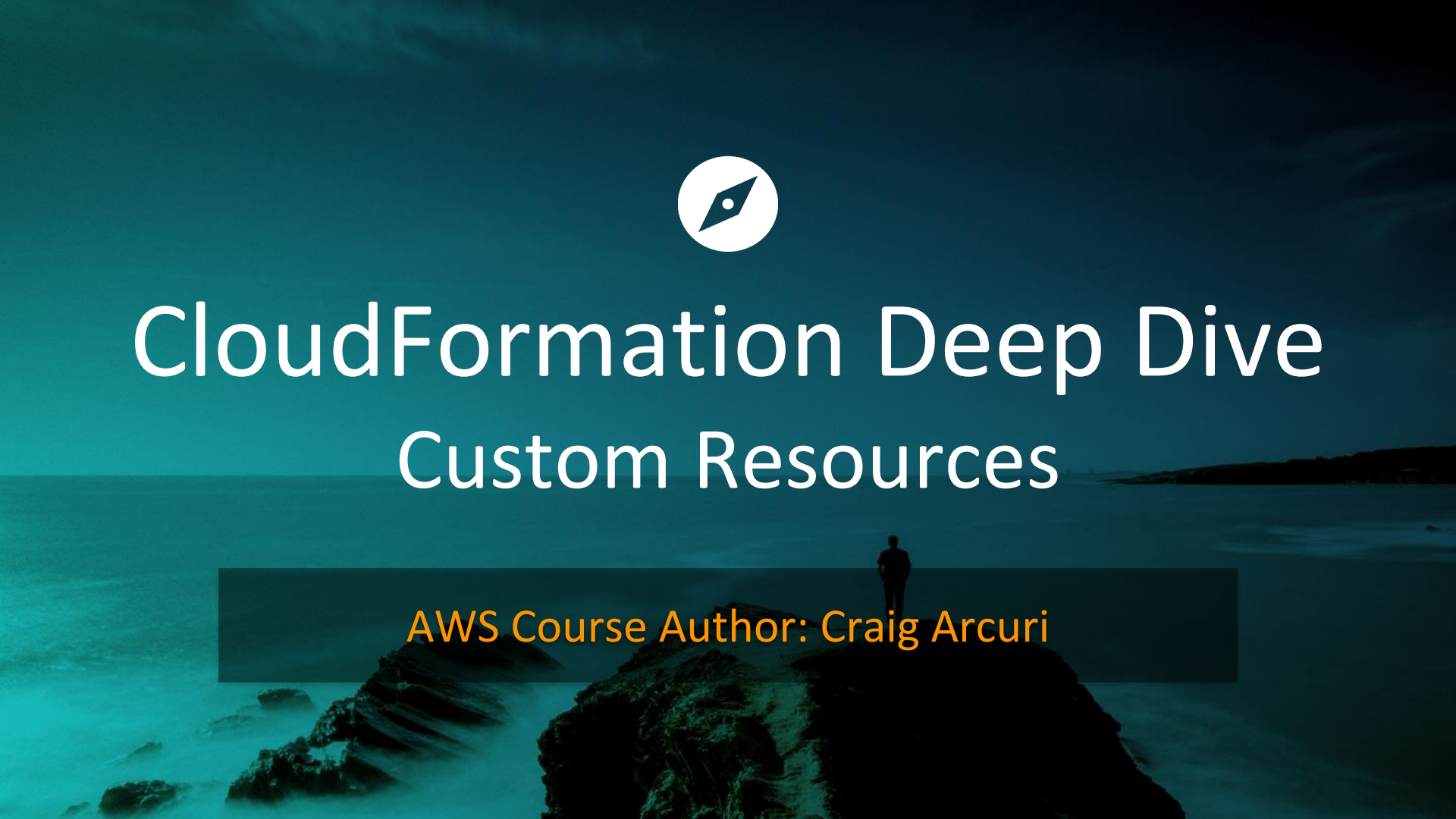
- The order of stack creation matters. Create the stack with the export first!

Nested Stacks

- What are Nested Stacks? Stacks created as part of other stacks.
- AWS::CloudFormation::Stack – allows creation of a stack within another stack.
- Code reuse, modularization.
- Common configurations: Load Balancers are a good example.
- Nested stacks can contain nested stacks.
- Stack updates should be initiated from the root stack.

Cross-Stack References vs Nested Stacks

- Nested Stacks – Deploy and manage all stacks from one stacks
- Cross-Stack References – Manage all stacks separately
- Need to isolate sharing of information? Use Nested Stacks
- Share information with other stacks? Use Cross-Stack References.



CloudFormation Deep Dive

Custom Resources

AWS Course Author: Craig Arcuri

Custom Resources Defined

- Enable you to write custom provisioning logic in templates that CloudFormation runs anytime you create, update, or delete stacks.
- You might want to include resources that aren't available as CloudFormation resource types.
- Use custom resources to include these resources and still manage all resources in a single stack.
- Use AWS::CloudFormation::CustomResources or Custom::String resource types to include custom resources in your template.
- One property required: Service token – specifies where CloudFormation sends requests to, such as SNS.

Custom Resources – How They Work

- Any action taken for a custom resource involves three parties:

  - template developer - Creates a template that includes a custom resource type. The template developer specifies the service token and any input data in the template.

  - custom resource provider - Owns the custom resource and determines how to handle and respond to requests from AWS CloudFormation. The custom resource provider must provide a service token that the template developer uses.

  - CloudFormation - During a stack operation, sends a request to a service token that is specified in the template, and then waits for a response before proceeding with the stack operation.

Custom Resources – Process

1 The template developer defines a custom resource in the template, which includes a service token and any input data parameters. The service token specifies where AWS CloudFormation sends requests to, such as an SNS topic ARN or to a Lambda function ARN.

2 Whenever the template is used to create, update, or delete a custom resource, CloudFormation sends a request to the specified service token.

Request Example:

```
{
  "RequestType" : "Create",
  "ResponseURL" : "http://pre-signed-S3-url-for-response",
  "StackId" : "arn:aws:cloudformation:us-west-2:EXAMPLE/stack-name/guid",
  "RequestId" : "unique id for this create request",
  "ResourceType" : "Custom::TestResource",
  "LogicalResourceId" : "MyTestResource",
  "ResourceProperties" : {
    "Name" : "Value",
    "List" : [ "1", "2", "3" ]
  }
}
```

Custom Resources – Process continued

3 The custom resource provider processes the AWS CloudFormation request and returns a response of SUCCESS or FAILED to the pre-signed url.

4 After getting a SUCCESS response, CloudFormation proceeds with the stack operation. If a FAILURE or no response is returned, the operation fails.

Any output data from the custom resource is stored in the pre-signed URL location. The template developer can retrieve that data by using the Fn:GetAtt function.

AWS::CloudFormation::CustomResource

- Custom resources provide a way for you to write custom provisioning logic in a CloudFormation template and have CloudFormation run it during a stack operation, such as when you create, update or delete a stack.
- Syntax:

Type: "Custom::String“ ← you can define your own name or use CustomResource

Version: "1.0"

Properties:

ServiceToken: String

... provider-defined properties ...

- Using your own resource type names helps you differentiate the types of custom resources in your stack.

AWS::CloudFormation::CustomResource

- Custom resources provide a way for you to write custom provisioning logic in a CloudFormation template and have CloudFormation run it during a stack operation, such as when you create, update or delete a stack.
- Syntax:

Type: "Custom::String“ ← you can define your own name or use CustomResource

Version: "1.0"

Properties:

ServiceToken: String

... provider-defined properties ...

- Using your own resource type names helps you differentiate the types of custom resources in your stack.
- Only one property (ServiceToken) is defined for a Custom Resource.

AWS::CloudFormation::CustomResource

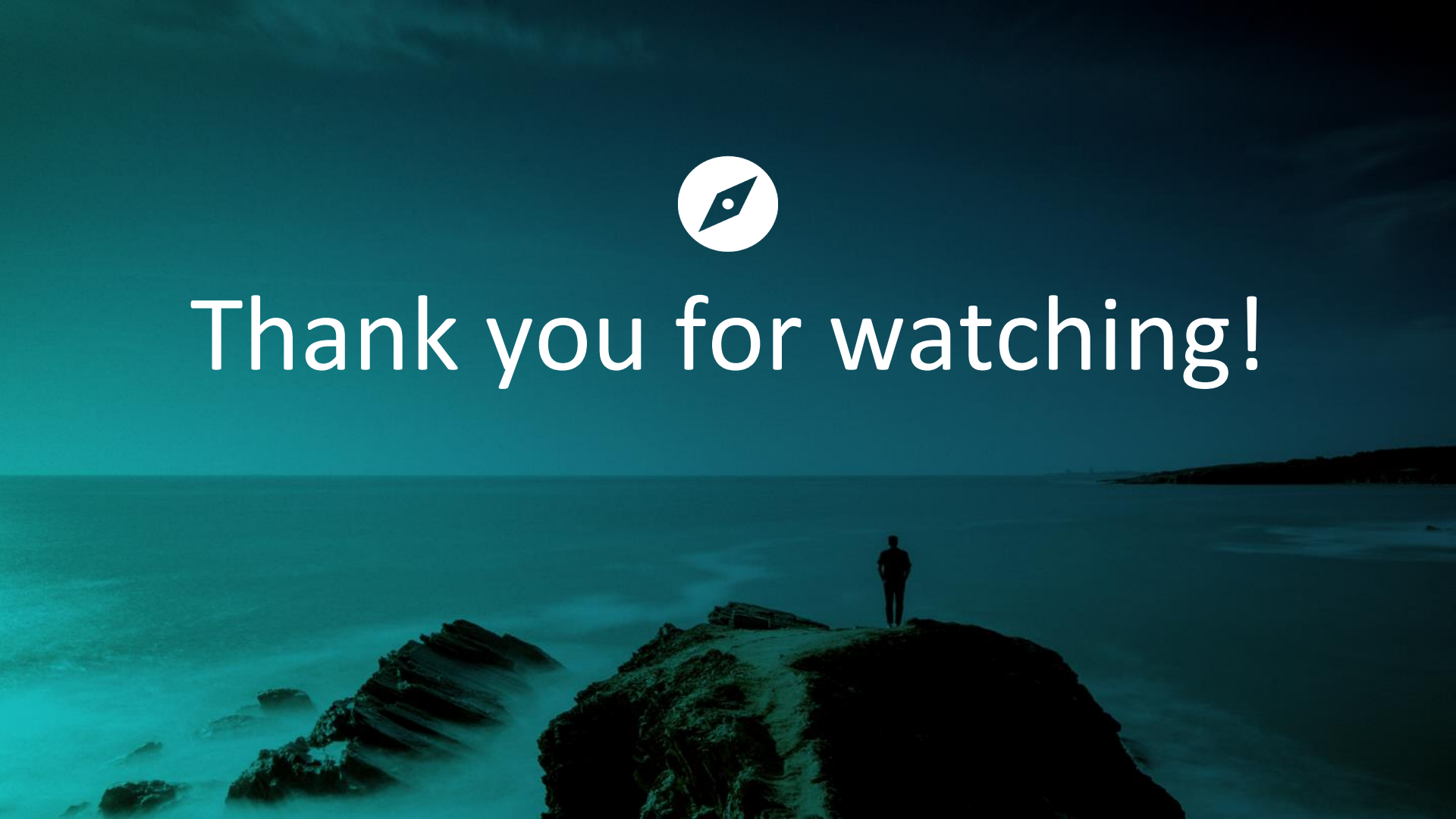
ServiceToken - The service token that was given to the template developer by the service provider to access the service, such as an SNS topic ARN or Lambda function ARN. The service token must be from the same region in which you are creating the stack.

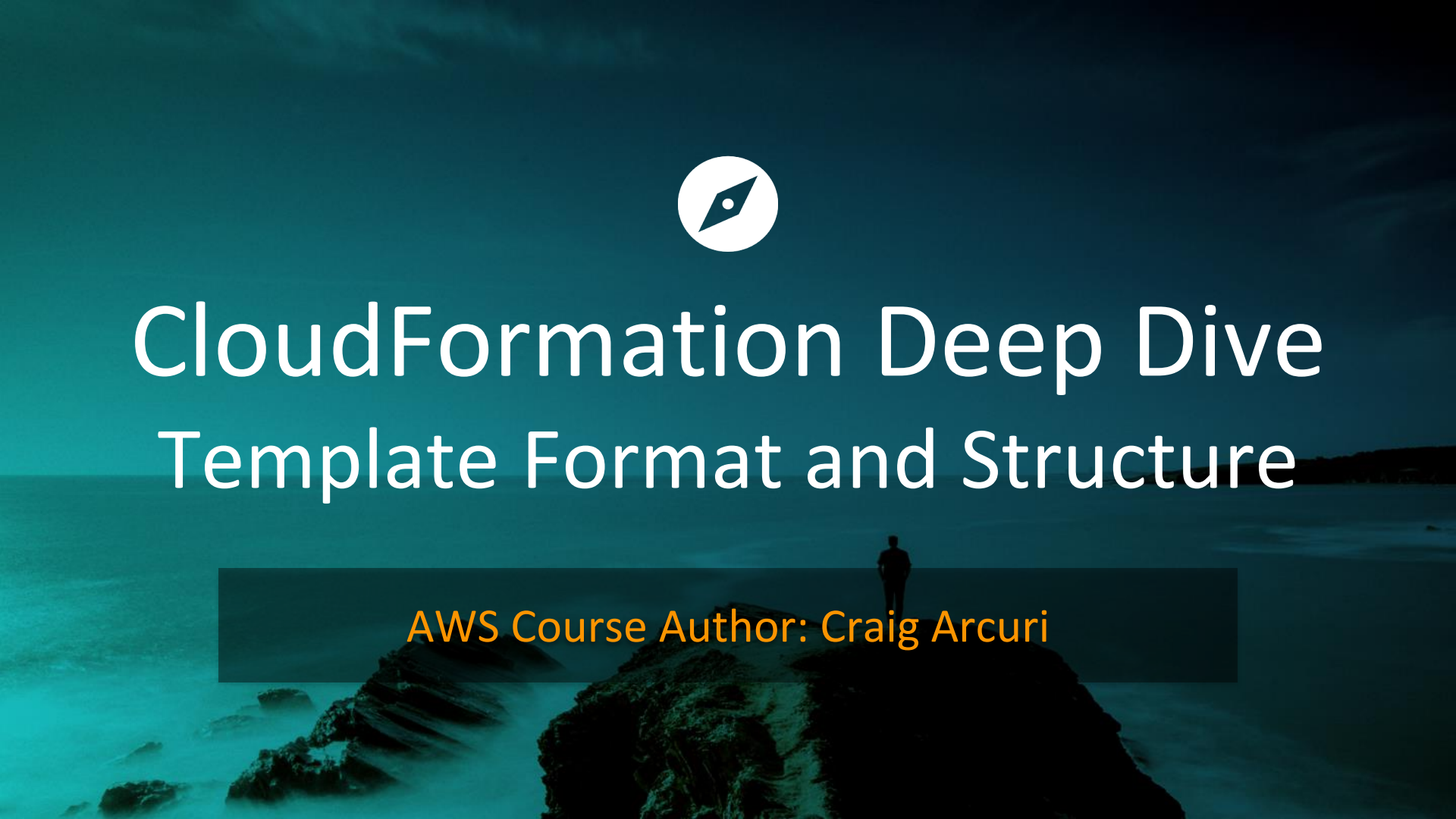
Return Values - defined by the custom resource provider, and are retrieved by and retrieved with Fn:GetAtt.

Custom Resource Definition – Lambda Example

```
MyCustomResource:
  Type: "Custom::TestLambdaCrossStackRef"
  Properties:
    ServiceToken:
      !Sub |

arn:aws:lambda:${AWS::Region}:${AWS::AccountId}:function:${LambdaFunctionName}
    StackName:
      Ref: "NetworkStackName"
```


Thank you for watching!



CloudFormation Deep Dive

Template Format and Structure

AWS Course Author: Craig Arcuri

Disaster Recovery with AWS

- Recovery Time Objective (RTO) - The time it takes after a disruption to restore a business process to its service level.
- Recovery Point Objective (RPO) - The acceptable amount of data loss measured in time. For example, if a disaster occurs at 12:00 PM (noon) and the RPO is one hour, the system should recover all data that was in the system before 11:00 AM. Data loss will span only one hour, between 11:00 AM and 12:00 PM (noon).

DR Scenarios with AWS – Backup and Restore

- Backup and Restore - Amazon S3 is an ideal destination for backup data that might be needed quickly to perform a restore.
- Steps:
  - Select an appropriate tool or method to back up your data into AWS.
  - Ensure that you have an appropriate retention policy for this data.
  - Ensure that appropriate security measures are in place for this data
  - Regularly test the recovery of this data and the restoration of your system.

DR Scenarios with AWS – Pilot Light

- Pilot Light - A minimal version of an environment is always running in the cloud. Typically have some preconfigured servers bundled as Amazon Machine Images (AMIs), which are ready to be started up at a moment's notice.
- Steps:
    - Start your application Amazon EC2 instances from your custom AMIs.
    - Resize existing database/data store instances to process the increased traffic.
    - Add additional database/data store instances to give the DR site resilience in the data tier; if you are using Amazon RDS, turn on Multi-AZ to improve resilience.
    - Change DNS to point at the Amazon EC2 servers.
    - Install and configure any non-AMI based systems, ideally in an automated way

DR Scenarios with AWS – Warm Standby

- A scaled-down version of a fully functional environment is always running in the cloud.
- Steps:
  - Increase the size of the Amazon EC2 fleets in service with the load balancer (horizontal scaling).
  - Start applications on larger Amazon EC2 instance types as needed (vertical scaling).
  - Either manually change the DNS records, or use Amazon Route 53 automated health checks so that all traffic is routed to the AWS environment.
  - Consider using Auto Scaling to right-size the fleet or accommodate the increased load.
  - Add resilience or scale up your database.

DR Scenarios with AWS – Multi-Site Solution

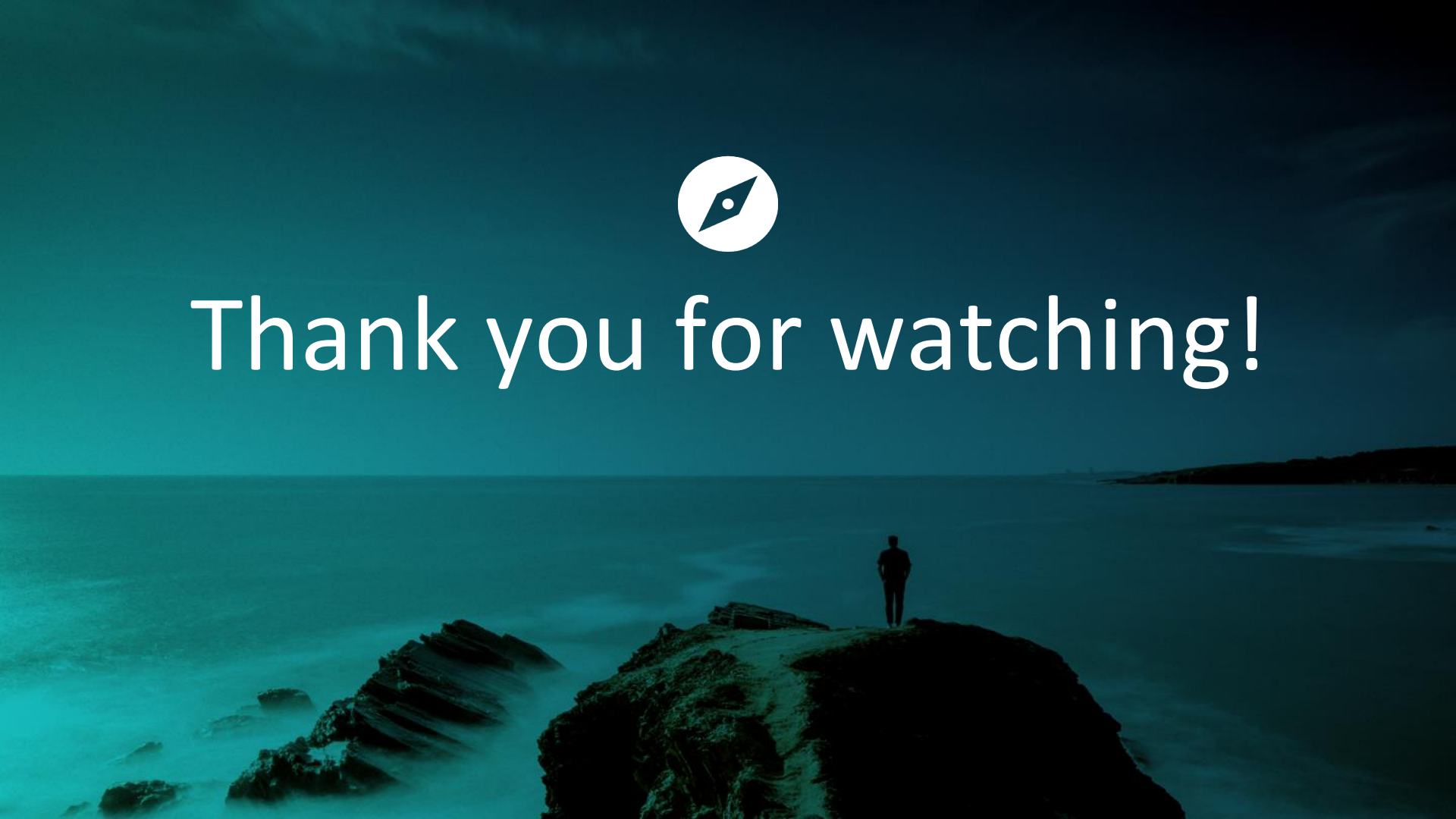
- Multi-site solution - runs in AWS as well as on your existing on-site infrastructure, in an active-active configuration. The data replication method that you employ will be determined by the recovery point that you choose.
- Steps:
    - Either manually or by using DNS failover, change the DNS weighting so that all requests are sent to the AWS site.
    - Have application logic for failover to use the local AWS database servers for all queries.
    - Consider using Auto Scaling to automatically right-size the AWS fleet.

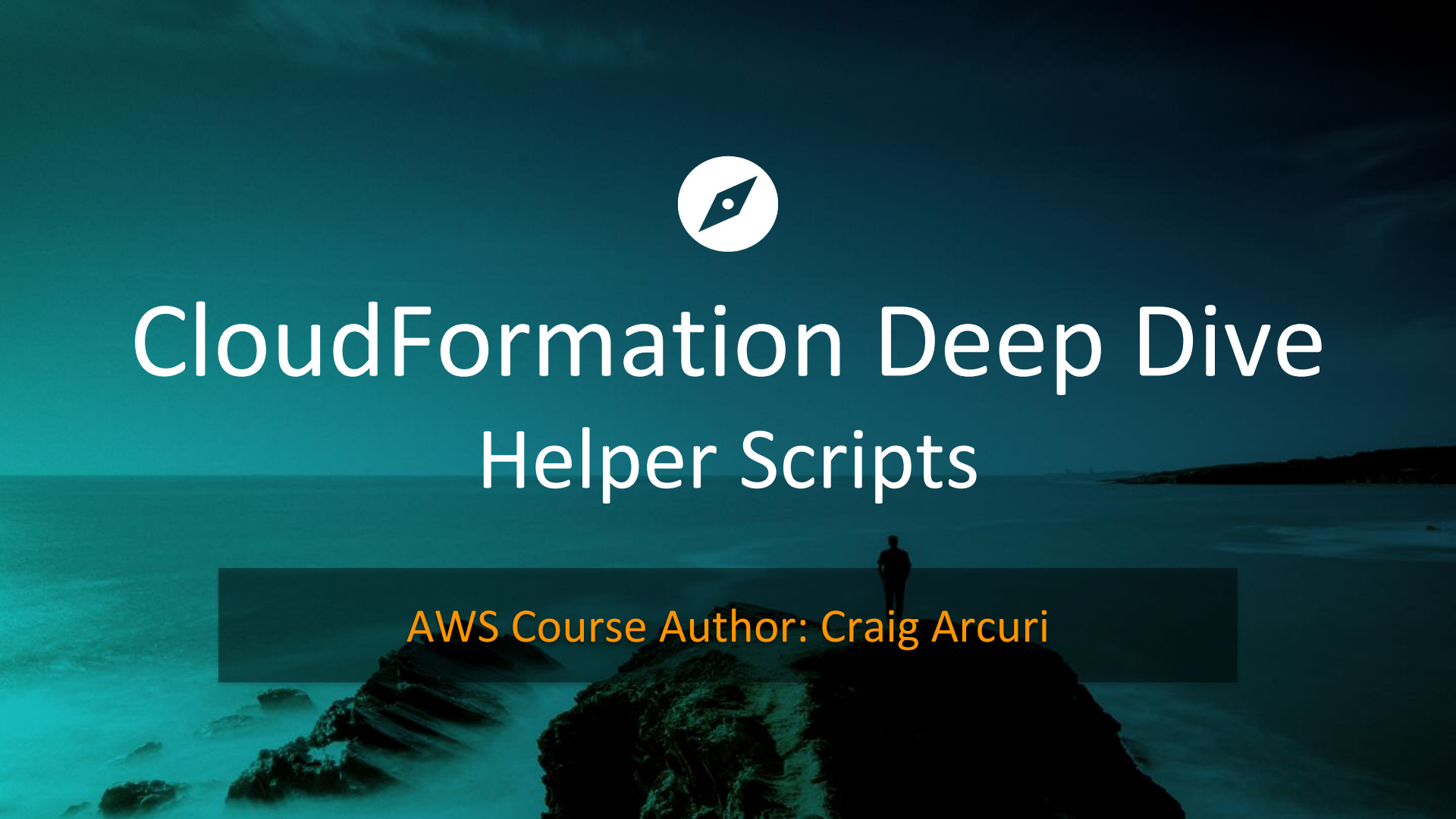
Disaster Recovery with CloudFormation

- With CloudFormation you can automate your recovery and meet more aggressive RTOs. Provisioning infrastructure services and building a resource stack with predefined templates provides a head start for your recovery.
- With CloudFormation, you can use templates to create stacks which provision EC2 and RDS instances as well as their respective settings and configurations in an organized and timely manner.
- Disaster Recovery Drills – Quickly spinup DR environments for testing and delete the stack when finished testing. Test a failover plan in a duplicate environment without impacting production resources.

Pilot Light with CloudFormation

- Web Server with RDS Database :
  - EC2 AMI of the Web Server
  - RDS Snapshot of the DB Server
  - CloudWatch Monitoring
  - AWS Lambda
  - CloudFormation (VPC, Subnet, IGW, Route Table, etc.)


Thank you for watching!



CloudFormation Deep Dive

Helper Scripts

AWS Course Author: Craig Arcuri

What Are Helper Scripts?

- CloudFormation provides a set of Python helper scripts that to install software and start services on EC2 instances created in your stack.
- You call the helper scripts directly from your template.
- The scripts work in with resource metadata that you define in your template.
- Helper scripts are pre-installed on the latest versions of the Amazon Linux AMI (in /opt/aws/bin).
- There are currently 4 helper scripts:
- cfn-init, cfn-signal, cfn-get-metadata, cfn-hup
- The scripts are not executed by default. You must include calls to execute specific helper scripts.

Use Cases

- Send signals back to a stack
- Configure and bootstrap instances
- Update instances

The Helper Scripts

- cfn-init - Used to retrieve and interpret the resource metadata, installing packages, creating files and starting services.
- cfn-signal - A wrapper to signal a CloudFormation CreationPolicy or WaitCondition, enabling you to synchronize other resources in the stack with the application being ready.
- cfn-get-metadata - A wrapper script making it easy to retrieve either all metadata defined for a resource or path to a specific key or subtree of the resource metadata.
- cfn-hup - A daemon to check for updates to metadata and execute custom hooks when the changes are detected.

cfn-init

- Reads template metadata from the AWS::CloudFormation::Init key
- Fetches and parses metadata from CloudFormation
- Installs packages
- Writes files to disk
- Enable/disable and start/stop services
- If you use cfn-init to update an existing file, it creates a backup copy of the original file in the same directory with a .bak extension.

cfn-init syntax

```
cfn-init --stack|-s stack.name.or.id \          ← required
         --resource|-r logical.resource.id \    ← required
         --region region
         --access-key access.key \
         --secret-key secret.key \
         --role rolename\
         --credential-file|-f credential.file \
         --configsets|-c config.sets \
         --url|-u service.url \
         --http-proxy HTTP.proxy \
         --https-proxy HTTPS.proxy \
         --verbose|-v
```

- Fetches and parses data from the specified stack and resource.

cfn-signal

- Signals CloudFormation to indicate whether Amazon EC2 instances have been successfully created or updated.
- Completed actions sends a success signal back. Failure signals will trigger a rollback.
- If you install and configure software applications on instances, you can signal CloudFormation when the applications are ready.
- Can use cfn-signal with a creation policy or an auto scaling group with a WaitOnResourceSignals update policy.
- When CloudFormation creates or updates resources with those policies, it suspends work on the stack until the resource receives the requisite number of signals or until the timeout period is exceeded.
- For each valid signal CloudFormation receives, it publishes the signals to the stack events so that you track each signal.

cfn-signal – resource signaling syntax

```
cfn-signal --success|-s signal.to.send \
       --access-key access.key \
       --credential-file|-f credential.file \
       --exit-code|-e exit.code \
       --http-proxy HTTP.proxy \
       --https-proxy HTTPS.proxy \
       --id|-i unique.id \
       --region AWS.region \
       --resource resource.logical.ID \        ← required
       --role IAM.role.name \
       --secret-key secret.key \
       --stack stack.name.or.stack.ID \        ← required
       --url AWS CloudFormation.endpoint
```

cfn-signal – with wait condition

```
cfn-signal --success|-s signal.to.send \
        --reason|-r resource.status.reason \
        --data|-d data \
        --id|-i unique.id \
        --exit-code|-e exit.code \
        waitconditionhandle.url      ← required
```

cfn-get-metadata

- Fetch a metadata block from CloudFormation and print it to standard out.
- Also print a sub-tree of the metadata block if you specify a key.

```
cfn-get-metadata --access-key access.key \
                 --secret-key secret.key \
                 --credential-file|f credential.file \
                 --key|k key \
                 --stack|-s stack.name.or.id \
                 --resource|-r logical.resource.id \
                 --url|-u service.url \
                 --region region
```

cfn-hup

- Detects changes in resource metadata and runs user-specified actions when a change is detected.
- Make configuration updates on running EC2 instances through the UpdateStack API action.

cfn-hup syntax

```
cfn-hup --config|-c config.dir \
        --no-daemon \
        --verbose|-v
```

--config – identify a config directory path. Default path is /etc/cfn/cfn-hup.conf (so it is not required)

--no-daemon – use this if you want to run the script only once rather than a scheduled daemon.

--verbose – returns details back from the command

cfn-hup conf file

[main]

stack=<stack-name-or-id>

credentials-file=<file>

region=<region>

interval=<number>

verbose=<boolean>

The stack name is required. Credentials are not required. Interval (in minutes) dictates how often to check for changes in metadata.

cfn-hup conf file - example

```
"files": {
    "/etc/cfn/cfn-hup.conf" : {
        "content" : { "Fn::Join" : ["", [
            "[main]\n",
            "stack=",  { "Ref" :  "AWS::StackId" }, "\n",
            "region=", { "Ref" : "AWS::Region::" }, "\n"
        ]}},
        "mode"   :    "000400",
        "owner" :     "root",
        "group"  :    "root"
```

- This goes in the template in AWS::CloudFormation::Init metadata section.

cfn-hup conf file - example

```
“services” : {
      “sysvinit” :  {
            “cfn-hup : { “enabled : “true”. “ensureRunning” : “true”,
                        “files” :  [ “/etc/cfn/cfn-hup.conf” ]}
            }
}
```

- Inside of the services key, enable hup and pass in the path of the config file.

cfn-hup hooks

How do we tell the daemon what actions to perform? The user actions that the cfn-hup daemon calls periodically are defined in the hooks.conf configuration file.

[hookname]

triggers=post.add or post.update or post.remove

path=Resources.<logicalResourceId> (.Metadata or .PhysicalResourceId)(.<optionalMetadatapath>)

action=<arbitrary shell command>

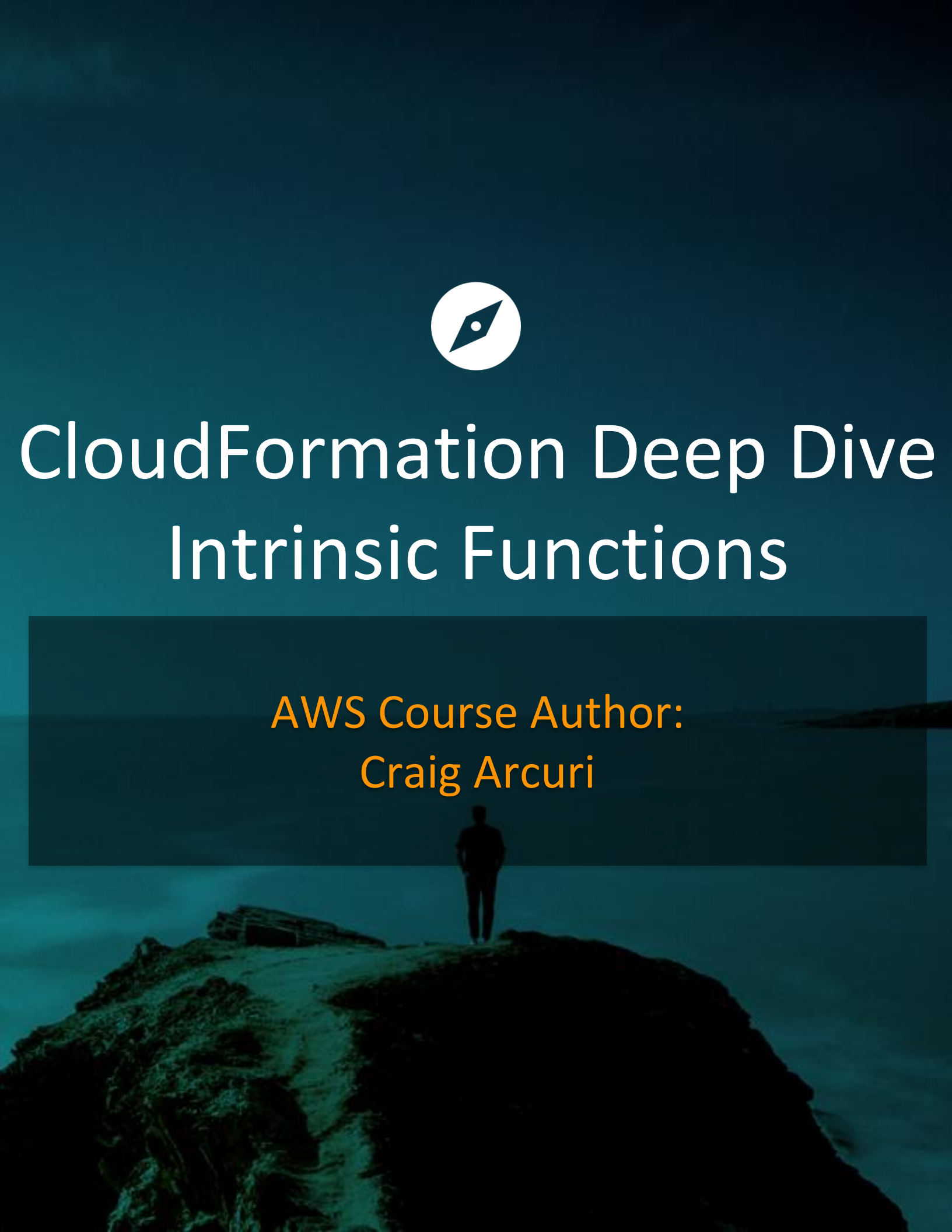
runas=<runas user>

cfn-hup hooks

- Triggers – list of conditions to detect.
- Path – represents the path of the metadata object
- Action – shell command to run if the hook is triggered
- Runas – which user to run the command as (root for example)

cfn-hup hooks – example (YAML)

```
...
  LaunchConfig:
    Type: "AWS::AutoScaling::LaunchConfiguration"
    Metadata:
      QBVersion: !Ref paramQBVersion
      AWS::CloudFormation::Init:
...
          /etc/cfn/hooks.d/cfn-auto-reloader.conf:
            content: !Sub |
              [cfn-auto-reloader-hook]
              triggers=post.update                    ← triggers on a post update
              path=Resources.LaunchConfig.Metadata.AWS::CloudFormation::Init
              action=/opt/aws/bin/cfn-init -v --stack ${AWS::StackName} --resource LaunchConfig --configsets wordpress_install --region ${AWS::Region}
              runas=root
            mode: "000400"
            owner: "root"
            group: "root"
...
```



CloudFormation Deep Dive Intrinsic Functions

AWS Course Author:
Craig Arcuri

Intrinsic Functions

- Built-in CloudFormation functions which allow you to dynamically assign values to properties at runtime.
- Example: Public IP address of an EC2 instance created in the stack.
- Help to manage CloudFormation stacks.
- Enables template reuse and portability.
- Intrinsic functions can only be used in specific parts of a template: resource properties, outputs, metadata attributes, and update policy attributes.
- Intrinsic functions can be used to conditionally create stacks resources.

Fn::Base64

- Sends the encoded data to EC2 instance from UserData property.

- JSON: { "Fn::Base64" : valueToEncode }

- YAML: Fn::Base64: valueToEncode

FindInMap

- Returns the value of key in two level map declared in Mappings section.
- JSON:
- { "Fn::FindInMap" : ["MapName", "TopLevelKey", "SecondLevelKey"] }
- YAML:
- Fn::FindInMap: [MapName, TopLevelKey, SecondLevelKey]

Fn::GetAtt

- Retrieves the value of an attribute from a resource in the template.
- Useful if you need to get the value at runtime.
- Examples – getting the public ip address of an EC2 instance created in the template. Getting the DNS name of an Elastic Load Balancer.
- JSON:
- { "Fn::GetAtt" : ["logicalNameOfResource", "attributeName"] }
- YAML:
- Fn::GetAtt: [logicalNameOfResource, attributeName]

Example: "Fn::GetAtt" : ["MyELB" , "DNSName"]

Fn::GetAzs

- Returns an array that lists Availability Zones for a specified region.
- Helps promote portable/reusable templates.
- AZs are not the same for every account. Hard coding is not best practice, but hard coding AZs could need changed often.
- JSON:
- { "Fn::GetAZs" : { "Ref" : "AWS::Region" } }

- YAML:
- Fn::GetAZs:

  Ref: "AWS::Region"

Fn::Join

- Used to append a list of values into a single value separated by provided delimiter.
- If a delimiter is the empty string, the set of values are concatenated with no delimiter.
- JSON:
- { "Fn::Join" : ["delimiter", [comma-delimited list of values]] }
- YAML:
- Fn::Join: [delimiter, [comma-delimited list of values]]
- Example (returns “a:b:c”):
- !Join [":", [a, b, c]]

Fn::Select

- Used to retrieve a single value from a given list. This function doesn't check index value. If given index is out of range then stack may get an error.
- JSON:
- { "Fn::Select" : [index, listOfObjects] }
- YAML:
- Fn::Select: [index, listOfObjects]
- **Example (returns 1)**

{ "Fn::Select" : ["2", ["0", "5", "1", "9", "3"]] }

- **Example for Azs:**

```
AvailabilityZone: !Select
        - 0
        - Fn::GetAZs: !Ref 'AWS::Region'
```

Fn::Split

- Splits a given string by delimiter and returns a list.
- JSON:
- { "Fn::Split": ["delimiter", "string"] }
- YAML:
- Fn::Split: ["delimiter", "string"]

- Example:
- { "Fn::Split" : ["|" , "a|b|c"] }
- Returns “a”, “b”, “c”

Fn::Sub

- Substitutes the variable in an input string with the specified values.
- Use this function to construct commands or outputs that include values that aren't available until you create or update a stack.
- JSON:
- { "Fn::Sub" : [String, { Var1Name: Var1Value, Var2Name: Var2Value }] }

{ "Fn::Sub": ["www.${Domain}", { "Domain": {"Ref" : "RootDomainName" }}]}

Ref

- Used to retrieve the value of given parameter or resource.
- JSON:
- { "Ref": "logicalname" }
- YAML:
- Ref: logicalname

```
"MyEIP" : {
  "Type" : "AWS::EC2::EIP",
      "Properties" : {
        "InstanceId" : { "Ref" : "MyEC2Instance" }
      }
}
```


Thank you for watching!

CloudFormation Deep Dive

Update Policy

AWS Course Author: Craig Arcuri

What is Lambda?

- Fully managed compute platform
- Run code without servers
- Executes on AWS infrastructure
- Event-Driven – create a Lambda function and configure it to respond to an event (S3 upload).
- Stateless
- Lambda removes all infrastructure complexity and responsibility

Lambda Administration

- Completely automated
- Highly available and fault tolerant
- Scales based on demand
- Automatic updates and patches to the OS
- Built in logging and monitoring

Lambda Essentials

- Lambda allocates:
  - CPU Power
  - Network Bandwidth
  - Disk I/O
- You allocate:
  - Memory
  - Execution timeout
- Lambda is stateless – need another service, such as DynamoDB to store state.

Lambda Essentials

- Supported Languages: Node.js, Java, Python
- Supported AWS Services:
    - DynamoDB
    - Kinesis
    - S3
    - SNS
- Events in these services can trigger Lambda functions.

Programming Concepts

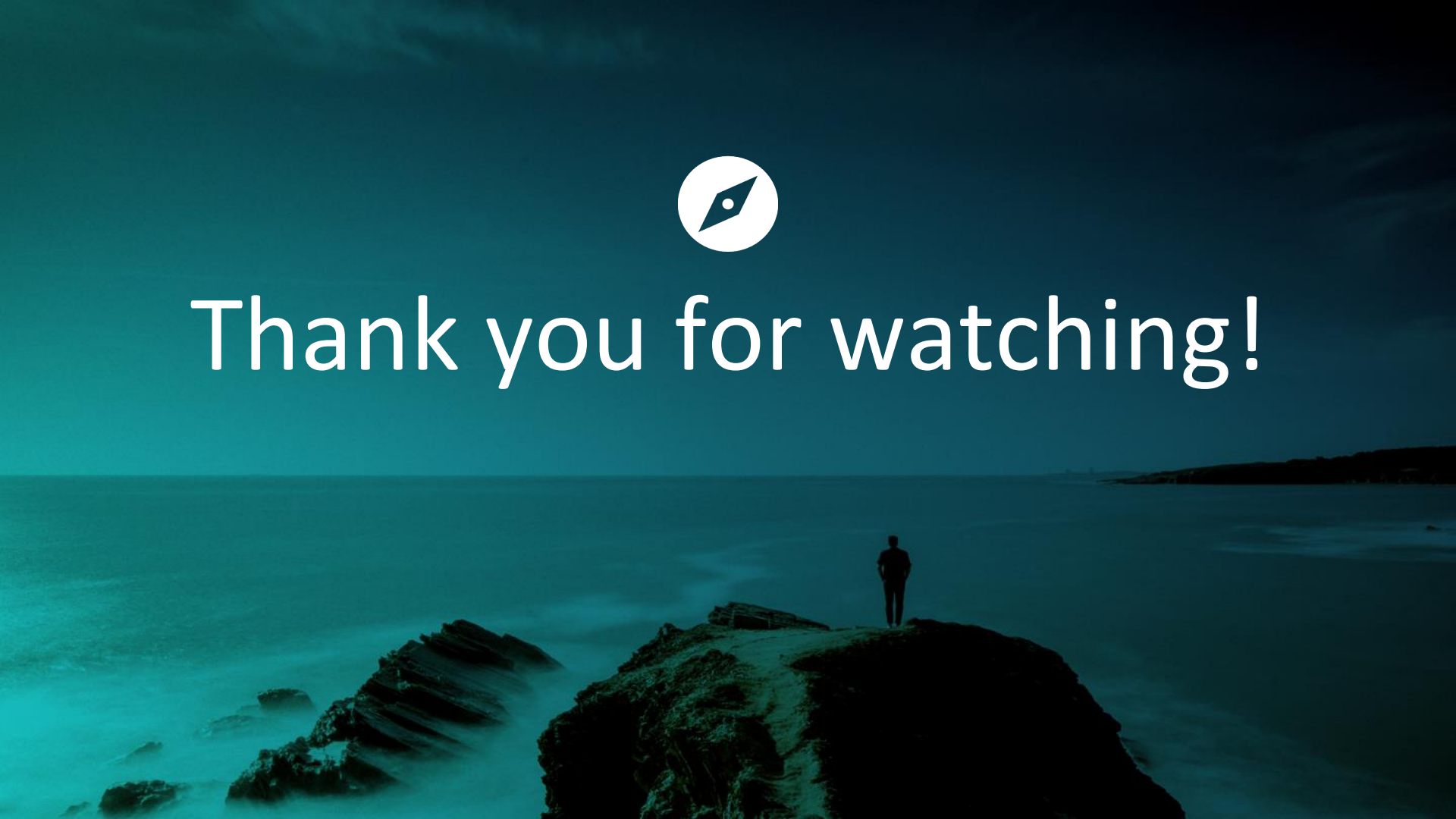
- Handler – The function called when the Lambda function is invoked.
- The handler processes event data.
- Handler has 2 parameters: Event – contains all event data, Context – passed into the handler as a second parameter and enables code to interact with Lambda.

Exceptions in Lambda

- Asynchronous Exception – Does not return a result to the caller. Logs error to CloudWatch Logs.
- Synchronous Exception – Returns result to the caller.

Event Driven Concepts

- Lambda code executes when driven by an event.
- Requirements for event driven execution:
  - A function to execute
  - An event source
  - Correct permission
- Event source examples – S3, DynamoDB, Amazon Echo, CLoudWatch Logs, CloudFormation, etc.


Thank you for watching!

CloudFormation Deep Dive StackSets

AWS Course Author:
Craig Arcuri

Introduction to StackSets

- Extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation.
- Use an Administrator account.
- Define and manage a CloudFormation template.
- Use the template to provision stacks into selected target accounts across specified regions.

StackSets Anatomy

- Work with Stack Sets, Stacks, and Stack Instances.
- Create and manage Stack Sets from an Administrator Account.
- A *target account* is the account into which you create, update, or delete one or more stacks in your stack set.
- Before you can use a stack set to create stacks in a target account, you must set up a trust relationship between the administrator and target accounts.

Stack Sets

- Lets you create stacks in AWS accounts across regions by using a single AWS CloudFormation template.
- All the resources included in each stack are defined by the stack set's AWS CloudFormation template.
- As you create the stack set, you specify the template to use, as well as any parameters and capabilities that template requires.
- After you've defined a stack set, you can create, update, or delete stacks in the target accounts and regions you specify.
- You can specify operation preferences, such as the order of regions in which you want the operation to be performed, the failure tolerance beyond which stack operations stop, and the number of accounts in which operations are performed on stacks concurrently.
- A stack set is a regional resource. If you create a stack set in one region, you cannot see it or change it in other regions.

Stack Instances

- A *stack instance* is a reference to a stack in a target account within a region.
- A stack instance can exist without a stack;
- if the stack could not be created for some reason, the stack instance shows the reason for stack creation failure.
- A stack instance is associated with only one stack set.

Stack Set Operations

Create Stack Set:

- Specify the template that you want to create the stacks.
- Specify the target accounts in which you want to create stacks.
- Identify the AWS regions in which you want to deploy stacks in your target accounts.

- A stack set ensures consistent deployment of the same stack resources, with the same settings, to all specified target accounts within the regions you choose.

Stack Set Operations - continued

Update Stack Sets:

- Push changes out to stacks in your stack set.
- Updates to the template always affects all stacks.
- you can’t selectively update the template for some stacks in the stacks set, but not others.
- Ways to update a stack set:
- Change existing settings in the template or add new resources, such as updating parameter settings for a specific service, or adding new Amazon EC2 instances.
- Replace the template with a different template.
- Add stacks in existing or additional target accounts, across existing or additional regions.

Stack Set Operations - continued

Delete Stacks: Remove a stack and all its associated resources from the target accounts you specify, within the regions you specify.

Ways to Delete Stacks:

- Delete stacks from some target accounts, while leaving other stacks in other target accounts running.
- Delete stacks from some regions, while leaving stacks in other regions running.
- Delete stacks from your stack set, but save them so they continue to run independently of your stack set by choosing the **Retain Stacks** option.
- Delete all stacks in your stack set, in preparation for deleting your entire stack set.

Delete Stack Sets: You can delete your stack set only when there are no stack instances in it.

Operations Options

- Control the time and number of failures allowed to successfully perform stack set operations.
- Prevent you from losing stack resources.

Maximum concurrent accounts:

- lets you specify the maximum number or percentage of target accounts in which an operation is performed at one time.
- Operations are performed in one region at a time, in the order specified in the **Deployment order** box.

Operations Options - continued

Fault Tolerance:

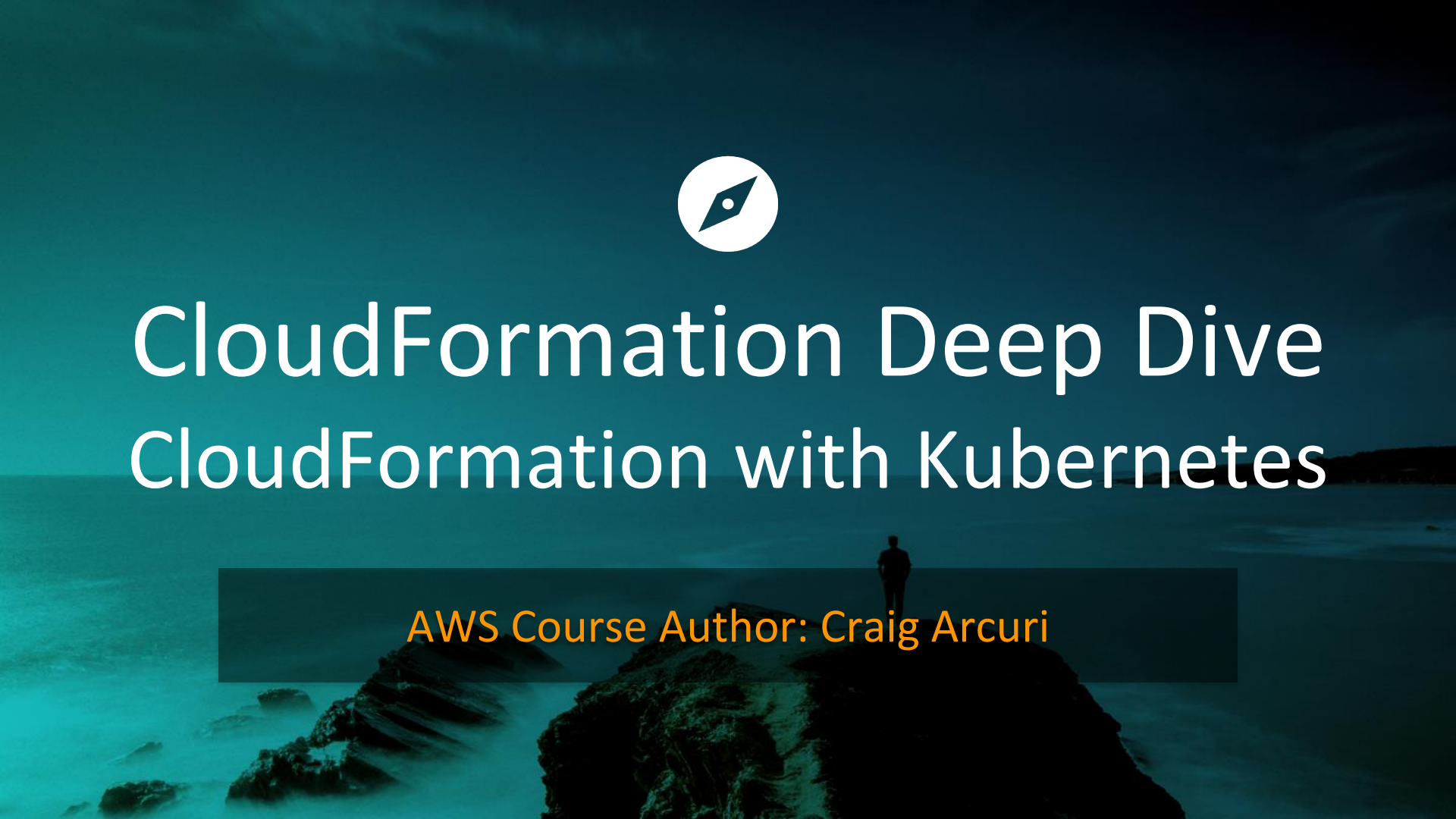
- Lets you specify the maximum number or percentage of stack operation failures that can occur per region.
- A lower number or percentage means that the operation is performed on fewer stacks, but you are able to start troubleshooting failed operations faster.
- if you are updating 10 stacks in 10 target accounts within three regions, setting **Failure tolerance** to **20** and **By percentage** means that a maximum of two stack updates in a region can fail for the operation to continue.

Operations Options - continued

Retain Stacks:

- Only available in delete stack workflows.
- lets you keep stacks and their resources running even after they have been removed from a stack set.
- Stacks are disassociated from the stack set, but the stack and its resources are saved.
- After a delete stacks operation is complete, you manage retained stacks in AWS CloudFormation, in the target account (not the administrator account) in which they were created.
- Retaining stacks permanently disassociates a stack from a stack set; the stack cannot be added to the stack set again, and it cannot be added to a new stack set.


Thank you for watching!



CloudFormation Deep Dive

CloudFormation with Kubernetes

AWS Course Author: Craig Arcuri

What is Kubernetes

- Kubernetes is an open-source platform designed to automate deploying, scaling, and operating application containers.
- Deploy your applications quickly and predictably.
- Scale your applications on the fly.
- Limit hardware usage to required resources only.
- **Portable**: public, private, hybrid, multi-cloud
- **Extensible**: modular, pluggable, hookable, composable
- **Self-healing**: auto-placement, auto-restart, auto-replication, auto-scaling

Why Containers?

- Deploy containers based on operating-system-level virtualization rather than hardware virtualization.
- These containers are isolated from each other and from the host: they have their own filesystems, they can't see each others' processes, and their computational resource usage can be bounded.
- They are easier to build than VMs, and because they are decoupled from the underlying infrastructure and from the host filesystem, they are portable across clouds and OS distributions.
- Because containers are small and fast, one application can be packed in each container image.
- With containers, immutable container images can be created at build/release time rather than deployment time, since each application doesn't need to be composed with the rest of the application stack, nor married to the production infrastructure environment. With containers, immutable container images can be created at build/release time rather than deployment time, since each application doesn't need to be composed with the rest of the application stack, nor married to the production infrastructure environment.

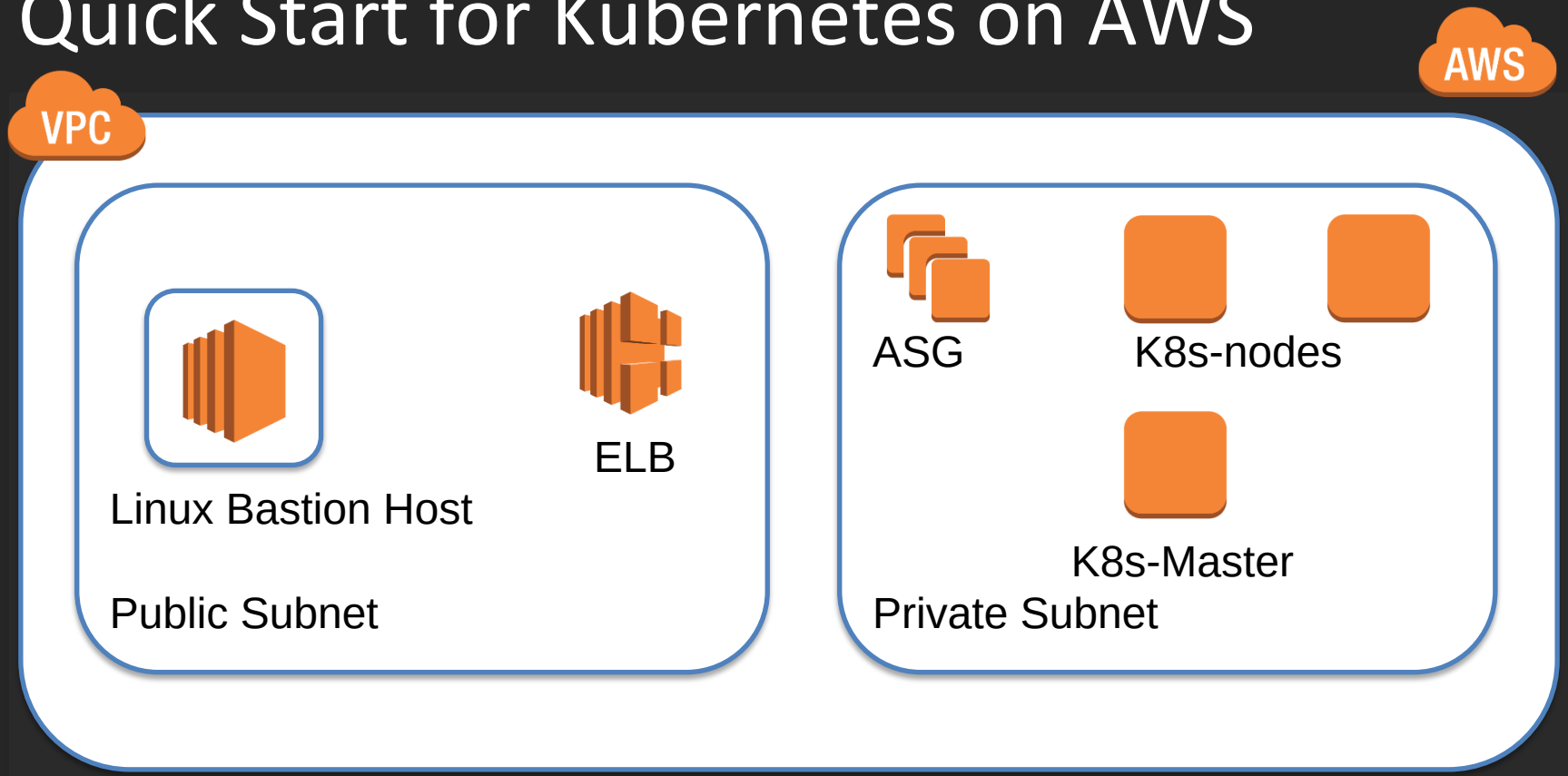
Benefits of Containers

- Generating container images at build/release time enables a consistent environment to be carried from development into production.
- Increased ease and efficiency of container image creation compared to VM image use.
- Create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- **Environmental consistency across development, testing, and production**
- **Cloud and OS distribution portability**: Runs on Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine, and anywhere else.
- Raises the level of abstraction from running an OS on virtual hardware to run an application on an OS using logical resources.

Quick Start for Kubernetes on AWS

- Gain the flexibility and security of the AWS infrastructure along with the powerful container deployment, scaling, and management capabilities of Kubernetes.
- The Quick Start for Kubernetes was created by AWS in partnership with Heptio.
- It sets up a flexible, secure AWS environment and launches a Kubernetes cluster automatically into a configuration of your choice.
    - VPC created by the Quick Start
    - Existing VPC

Quick Start for Kubernetes on AWS



Core Components

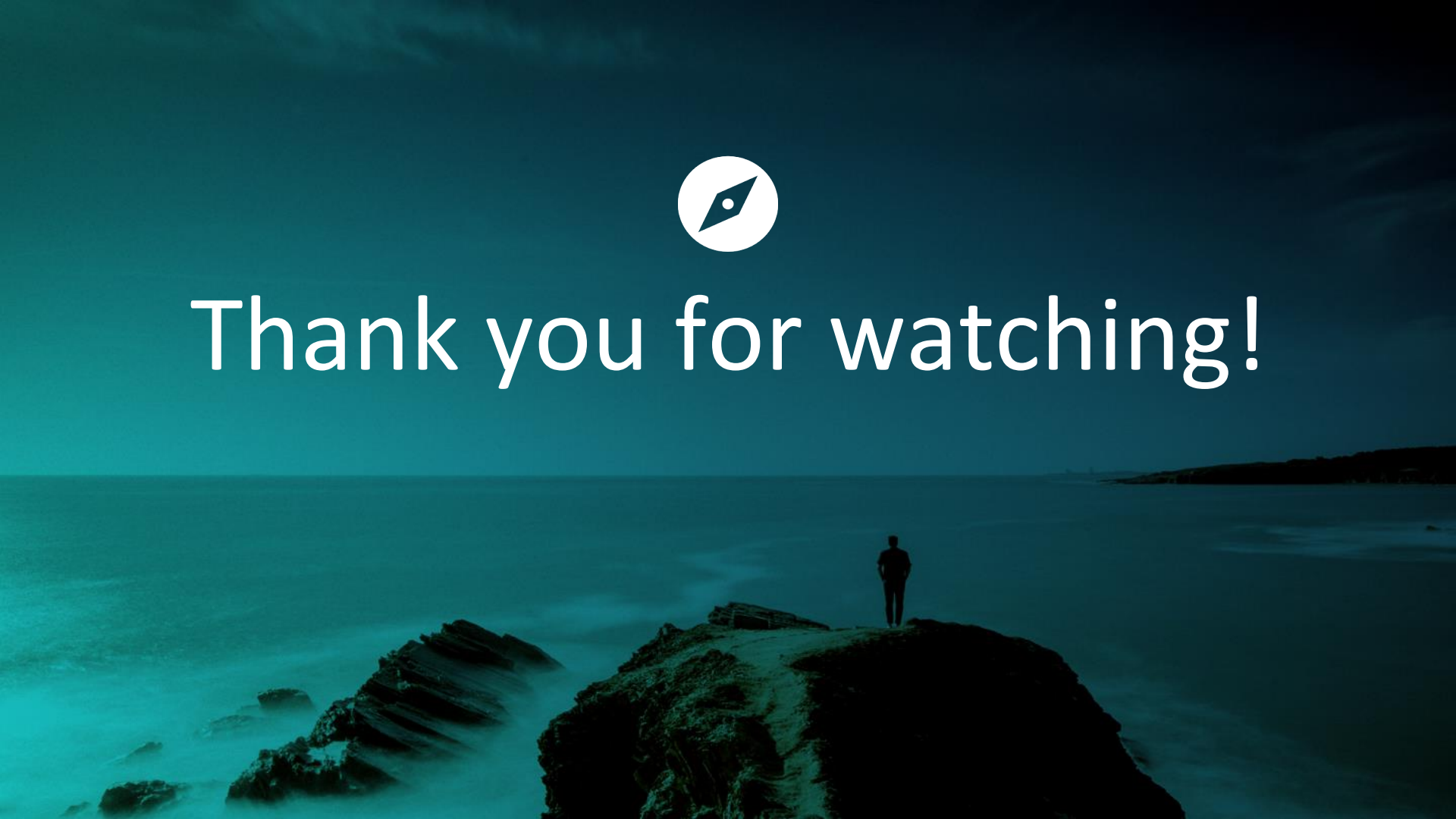
- A VPC that includes two subnets (public and private)
- In the public subnet, a Linux bastion host and Elastic Load Balancing
- In the private subnet, a Kubernetes cluster on Ubuntu 16.04 LTS. The Quick Start sets up one EC2 instance for the master node and your choice of 1-20 EC2 instances in an Auto Scaling group for worker nodes
- kubeadm for cluster administration, and Docker for the container runtime
- Your choice of Calico or Weave for networking between pods
- Security group for SSH access from the bastion host and inter-node connectivity

Quick Start for Kubernetes Overview

- Bootstraps your Kubernetes cluster with one master, two additional nodes by default, and a load balancer for HTTPS access to the Kubernetes API.
- You'll be able to learn how Kubernetes works at a manageable scale, with key parts in place for a full-scale deployment.

Kubernetes on AWS

- Orchestration software used for managing cloud workloads through containers (like Docker).
- Kubernetes helps assign containers to machines in a scalable way, keep them running in the face of failures and facilitating them talking to each other.
- AWS Cloud provides the infrastructure services your containerized workloads run on, while Kubernetes coordinates the containers in a flexible and fault-tolerant way.
- Kubernetes handles many of the details of traditional system administration and decouples workload deployment from infrastructure deployment.
- Kubernetes also integrates with AWS to utilize Amazon Elastic Block Store (Amazon EBS) volumes and expose services using Elastic Load Balancing.


Thank you for watching!