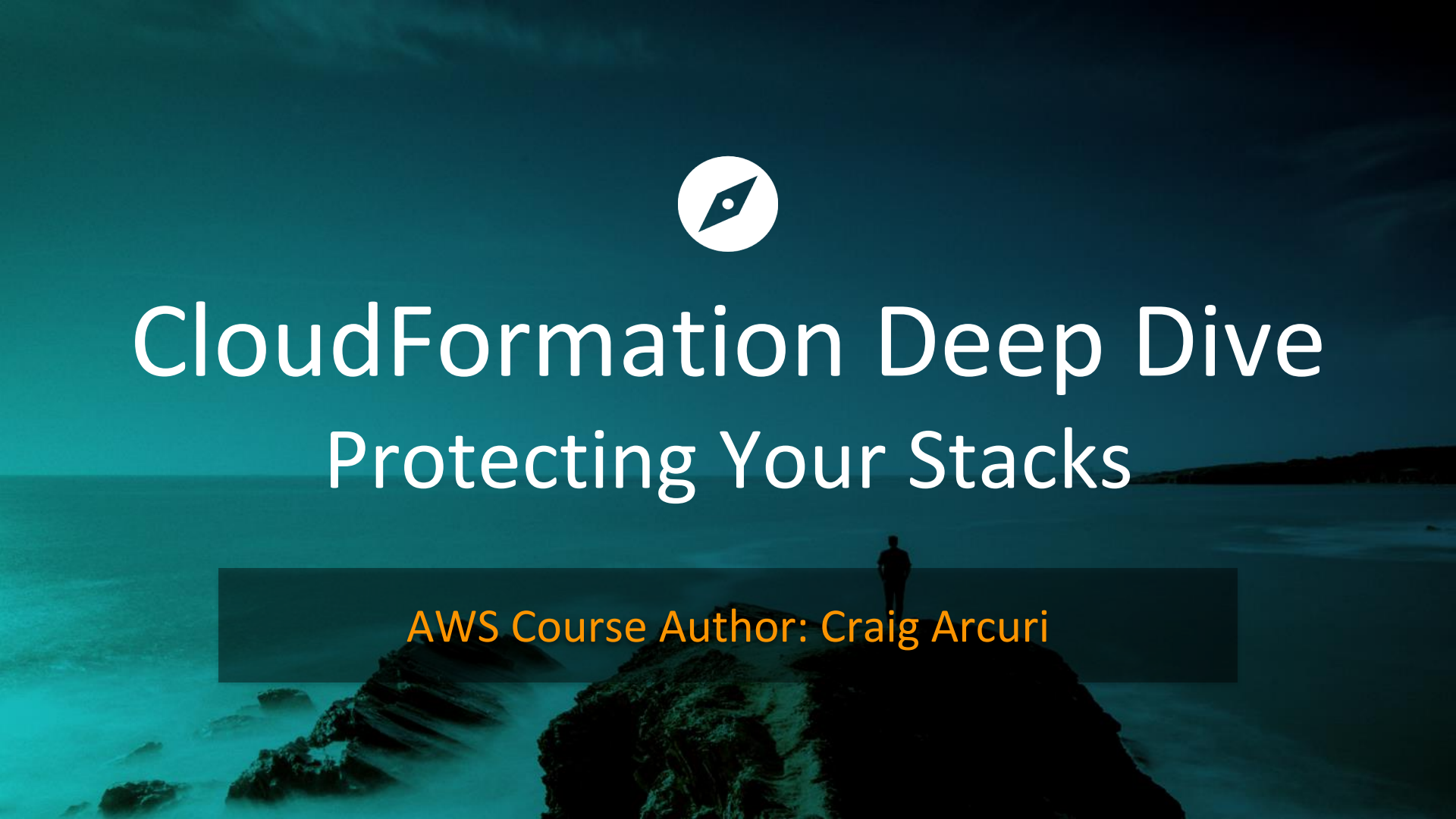


CloudFormation Deep Dive

Protecting Your Stacks

AWS Course Author: Craig Arcuri

Production Stacks need protection!

- You have production stacks, multiple architects with admin access to those stacks. How do you protect them from accidental deletion or update?
- Four Ways:

1 Termination Protection

2 Stack Level Policies

3 Resource Level Policies

4 IAM Policies

Termination Protection

- For stacks that hold critical resources you can apply termination protection.
- Enable termination protection while creating a new stack.
- Delete actions against the stack will be denied.
- Can enable termination protection via Management Console, CLI, and API.
- After you create your stack, termination protection in the **Overview** section of your stack.
- For nested stacks, termination protection cascades down to the sub-stacks (no need to manage termination protection individually for the sub-stacks).
- Termination protection can be removed but it is best practice to manage this privilege with IAM.

Stack Level Policies

- When you create a stack, all update actions are allowed on all resources.
- Stack policies can prevent accidental update or delete of stacks
- A stack policy is a JSON document that defines the update actions that can be performed on designated resources.
- After you set a stack policy, all of the resources in the stack are protected by default.
- Set an explicit allow on resources that you want to allow updates on.
- You can define only one stack policy per stack
- stack policy applies to all AWS CloudFormation users who attempt to update the stack.

Resource Level Policies

- Deletion Policy – Used to preserve or backup a resource when its stack is deleted.
- Specify a DeletionPolicy attribute for each resource that you want to control.
- A resource is deleted by default if there is no DeletionPolicy attribute.
- Use Retain to keep a resource after the stack is deleted.
- Specify snapshot (for resources that support snapshots) to have CloudFormation create a snapshot before deleting the resource.
- DeletionPolicy options – Delete (this is the default option), Retain, Snapshot

IAM Policies

- **AWS::IAM::Policy -** associates an IAM policy with IAM users, roles, or groups.
- JSON:

```
{
  "Type" : "AWS::IAM::Policy",
  "Properties" : {
    "Groups" : [ String, ... ],
    "PolicyDocument" : JSON object,
    "PolicyName" : String,
    "Roles" : [ String, ... ],
    "Users" : [ String, ... ]
  }
}
```

IAM Policies

- YAML:

```
Type: "AWS::IAM::Policy"
Properties:
  Groups:
    - String
  PolicyDocument: JSON object
  PolicyName: String
  Roles:
    - String
  Users:
    - String
```

IAM Policy Properties

- Groups - The names of groups to which you want to add the policy.
- PolicyDocument - contains permissions to add to the specified users or groups.
- PolicyName
- Roles - The names of AWS::IAM:Roles to which this policy will be attached.
- Users - The names of users for whom you want to add the policy.

IAM Policy Example

```
Type: "AWS::IAM::Policy"
Properties:
  PolicyName: "root"
  PolicyDocument:
    Version: "2012-10-17"
    Statement:
      -
        Effect: "Allow"
        Action: "*"
        Resource: "*"
  Roles:
    -
      Ref: "RootRole"
```

CloudFormation Deep Dive

Scaling Across Accounts and Regions

AWS Course Author: Craig Arcuri

The Problem

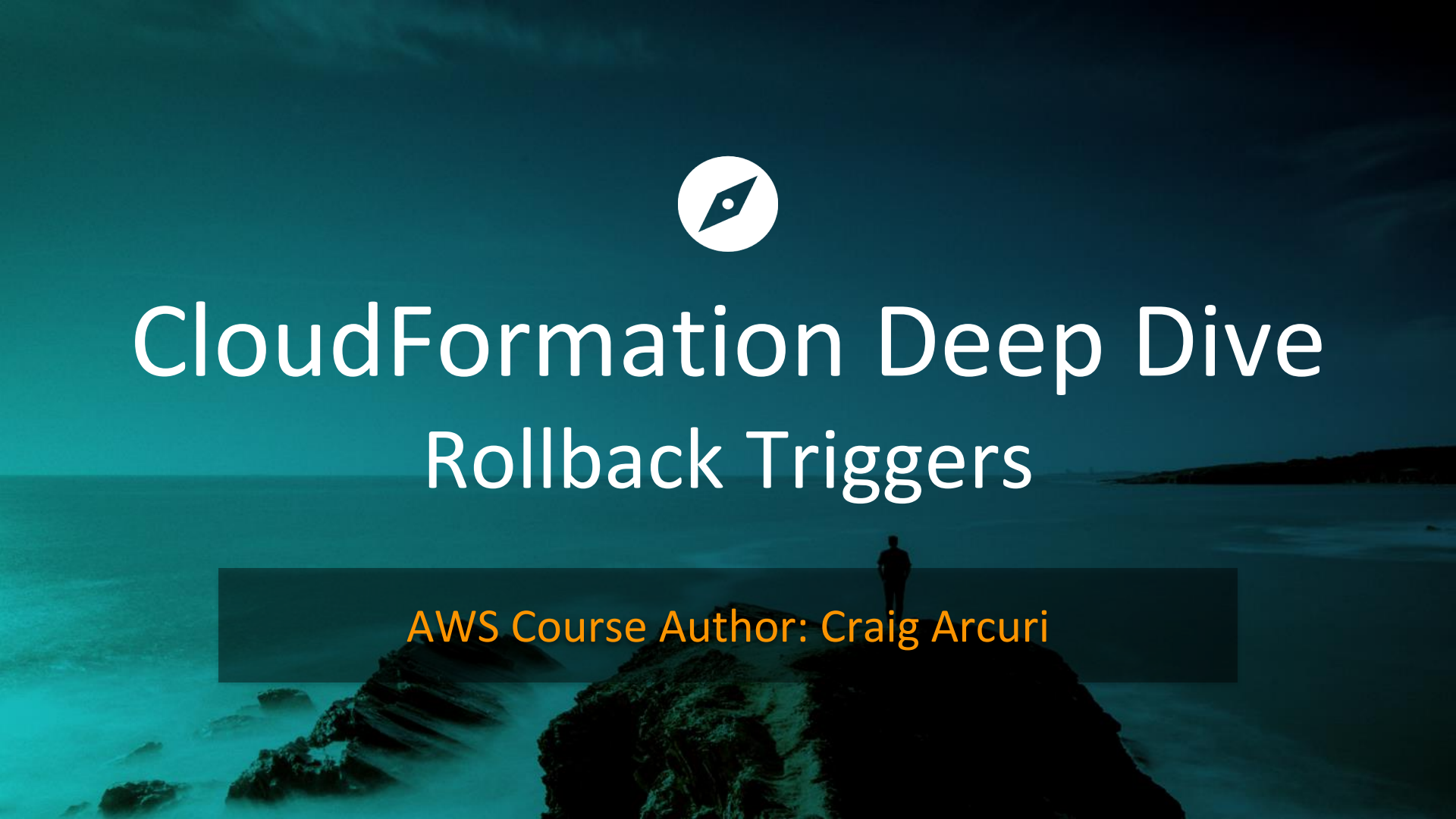
- Companies can span multiple accounts and regions.
- May have thousands of accounts in multiple regions.
- Smaller scale companies, maybe just 10s of accounts and a few regions. Same deployment problems though. Looking for consistent deployment process.
- High Availability can be attained by deploying in multiple regions.
- Multi-region may be selected for security or compliance reasons.

Issues With Multi Account and Region Deployments

- Manual operations in different accounts and regions can introduce errors and inconsistencies.
- Non-standard scripts in different regions add maintenance headaches.
- Third party tools introduce more complexity and cost.

Solution? Stack Sets!

- Use Stack Sets to standardize deployments across regions.
- Enable CloudTrail in all regions.
- Set up KMS keys
- Use AWS Config to tag resources
- Use CloudFormation Stack Sets to standardize and speed up deployments across regions.
- Use for DR – S3 Bucket replication, provision read replicas.
- Centralized Management
- Reduced complexity



CloudFormation Deep Dive

Rollback Triggers

AWS Course Author: Craig Arcuri

Monitor Your Stacks With Cloudwatch

- Integrate Alarms from CloudWatch with Cloudformation
- Monitor these alarms during updates to stacks.
- Rollback when the alarms fire.

Rollback Trigger Details

- Allows you to integrate application- and resource-level alarms from Amazon CloudWatch into the update process for your stacks.
- If a change to the stack causes any of the registered alarms to fire, CloudFormation immediately stops the update and rolls back to the last good state.
- You can include a monitoring window after all updates are complete to allow additional time for the change to stabilize.
- This window happens prior to the CloudFormation cleanup phase, allowing attribute changes and replaced resources to be quickly restored.

Rollback Trigger Details - continued

- When you create a rollback trigger, you need to specify the Cloudwatch alarm that CloudFormation should monitor.
- CloudFormation monitors the specified alarms during the stack create or update operation.
- You specify the amount of time after all resources have been deployed for the rollback triggers to be in effect.
- If an alarm goes to ALERT state during the stack operation or the monitoring period, CloudFormation rolls back the entire stack operation.
- If there are no alarms that go to ALERT state, CloudFormation proceeds to dispose of old resources as usual.

Rollback Trigger Details - continued

- CloudFormation, by default, only rolls back stack operations if an alarm goes to ALERT state, not INSUFFICIENT_DATA state.
- To have CloudWatch roll back if an alarm goes to INSUFFICIENT_DATA, edit the CloudWatch alarm to treat missing data as breaching.

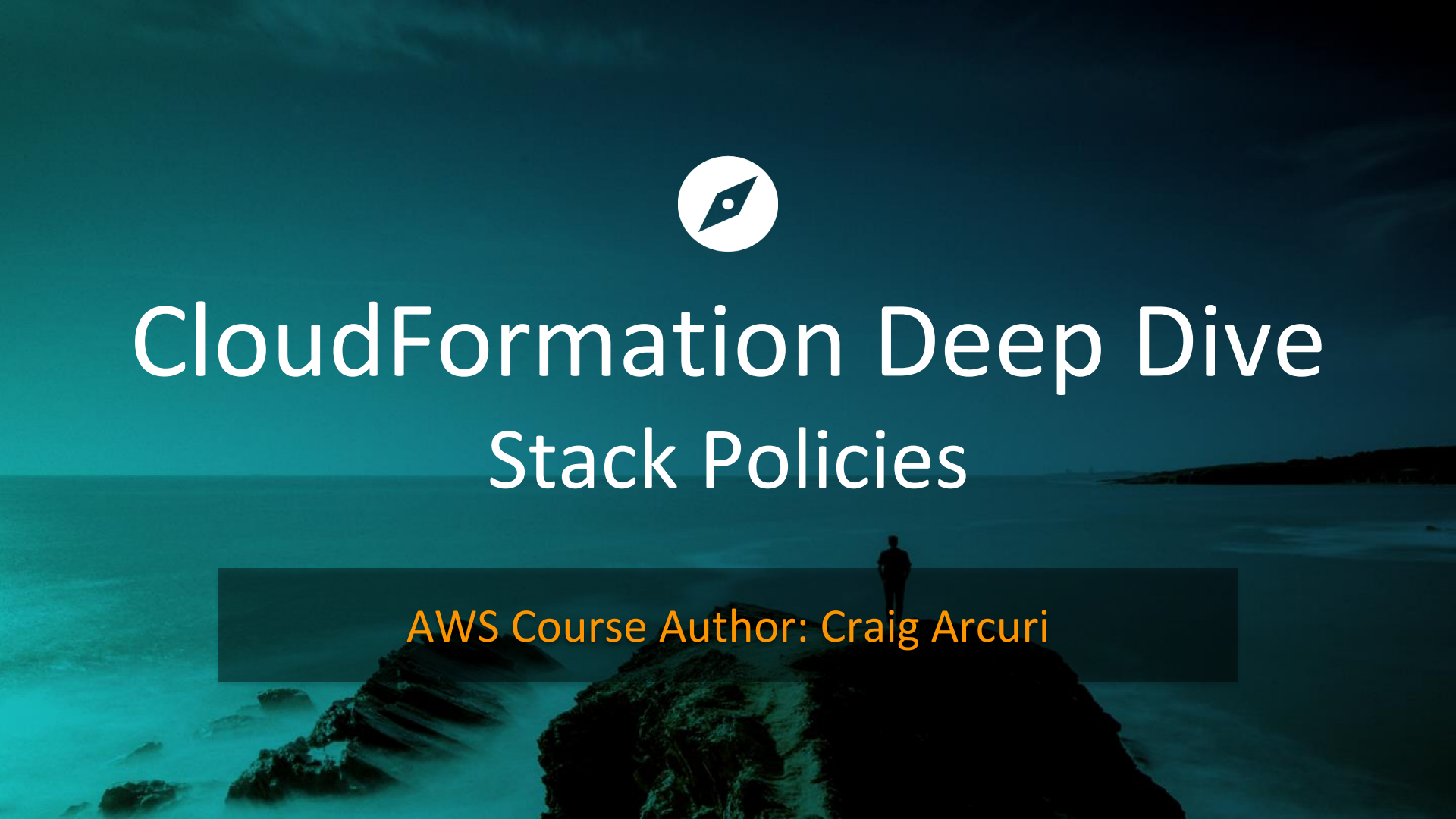
Rollback Trigger Parameters

- ARN - The Amazon Resource Name (ARN) of the rollback trigger.
- Type - The resource type of the rollback trigger (currently only AWS::CloudWatch:Alarm) is supported.

```
{
  “Rollback Triggers”:[
        {
         “Arn”: xxxxxxxxxxxxxxxxxxxxxxxxxxx,
         “Type”: “AWS::CloudWatch:Alarm”
        }
      ],
  “MonitoringTimeInMinutes”: 10
}
```

MonitoringTimeInMinutes

- The amount of time that CloudFormation should monitor all the rollback triggers after the stack creation or update operation deploys all necessary resources.
- If any of the alarms goes to ALERT state during the stack operation or this monitoring period, CloudFormation rolls back the entire stack operation.
- Updates - if the monitoring period expires without any alarms going to ALERT state CloudFormation proceeds to dispose of old resources as usual.
- If you specify a monitoring period but do not specify any rollback triggers, CloudFormation still waits the specified period of time before cleaning up old resources for update operations. This monitoring period can be used to perform manual stack validation.



CloudFormation Deep Dive

Stack Policies

AWS Course Author: Craig Arcuri

Stack Policies

- Remember, when you create a stack all update actions are allowed on all resources. By default, anyone with stack update permissions can update all of the resources in the stack.
- prevent stack resources from being unintentionally updated or deleted during a stack update by using a stack policy.
- JSON document that defines the update actions that can be performed on designated resources.

Stack Policy Syntax

```
{
  "Statement" : [
    {
      "Effect" : "Deny_or_Allow",
      "Action" : "update_actions",
      "Principal" : "*",
      "Resource" : "LogicalResourceId/resource_logical_ID",
      "Condition" : {
        "StringEquals_or_StringLike" : {
          "ResourceType" : [resource_type, ...]
        }
      }
    }
  ]
}
```

Stack Policy Example

```
{
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "*"
    },
    {
      "Effect" : "Deny",
      "Action" : "Update:*",
      "Principal": "*",
      "Resource" : "LogicalResourceId/ProductionDatabase"
    }
  ]
}
```

- Must explicitly allow an action on a resource.

Stack Policy Syntax

- Policies can be created at stack creation time, or applied to an existing stack.
- 1 stack policy per stack.
- Policy elements: Effect, Action, Principal, Resource, Condition

String Equals Condition

```
"Conditions" : {
  "StringEquals" : {
    "ResourceType" : ["AWS::DynamoDB::Table"]
```

- Any resource of this type will be subject to the Effect and Action applied to it.

String Equals Condition

```
“Conditions” : {
   “StringLike” : {
      “ResourceType” : [“AWS::EC2::*”]
```

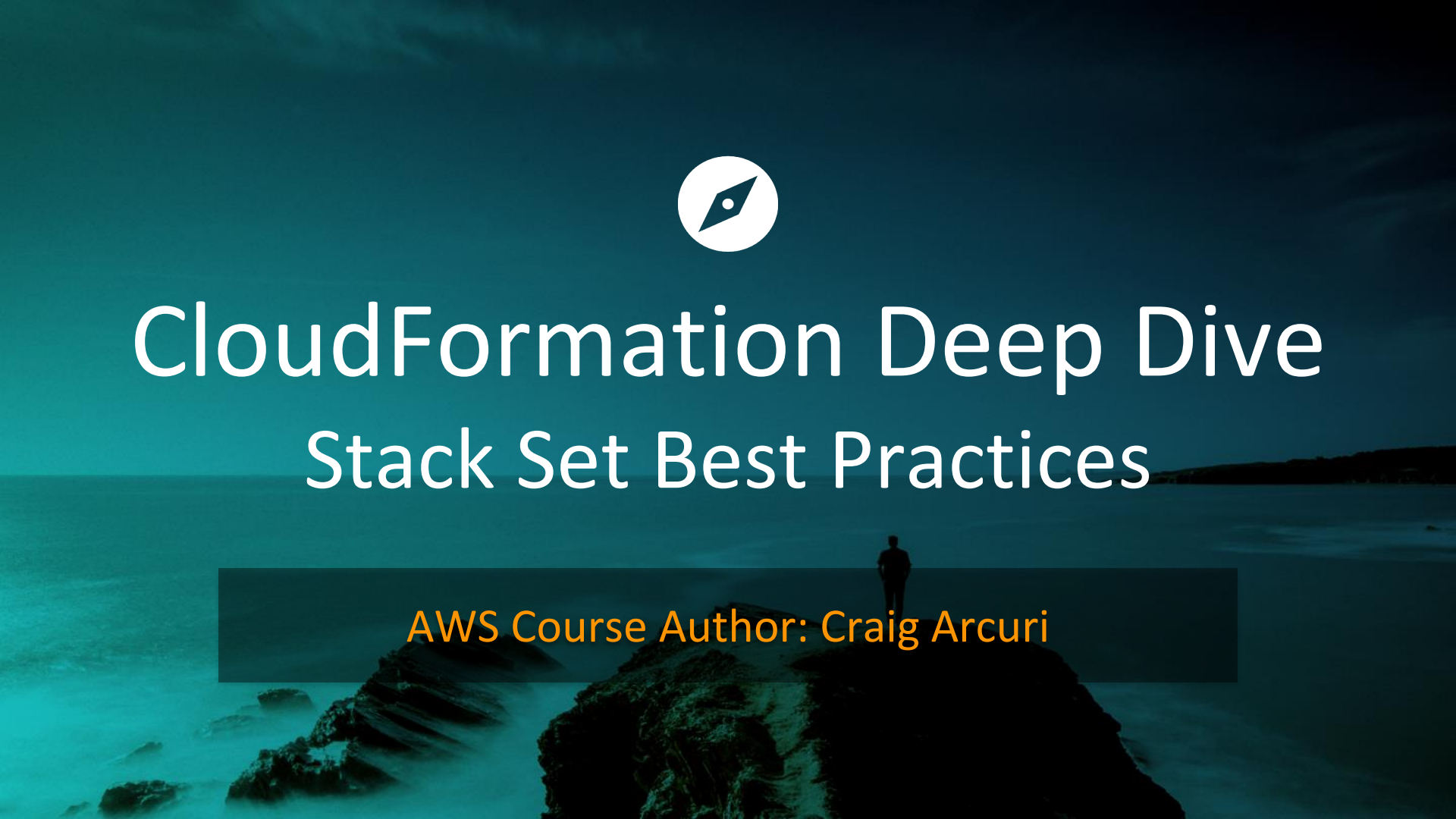
- Any resource like “EC2” will be subject to the Effect and Action applied to it.

Actions

- Update:Modify
- Update:Replace
- Update:Delete
- Update:*

Updating Protected Resources

- Overriding stack policy – temporary, only for the specific update. Specified at update time.
- The overriding policy needs to have an allow statement for the resources we want to update.



CloudFormation Deep Dive

Stack Set Best Practices

AWS Course Author: Craig Arcuri

Defining The Template

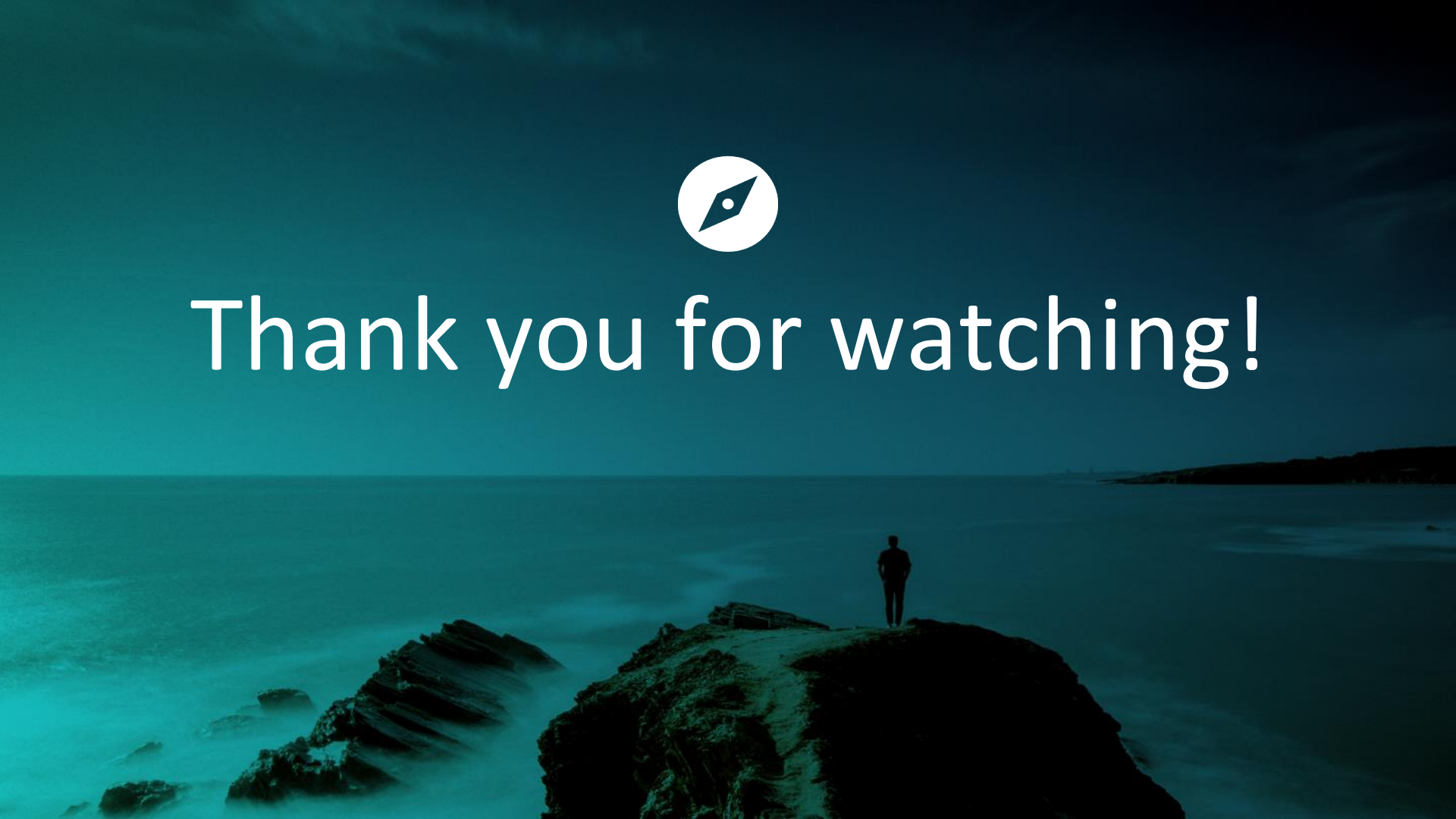
- Define the template that you want to standardize in multiple accounts, within multiple regions.
- Be sure that global resources (such as IAM roles and Amazon S3 buckets) do not have naming conflicts when they are created in more than one region in the same account.
- Store templates in S3 Buckets

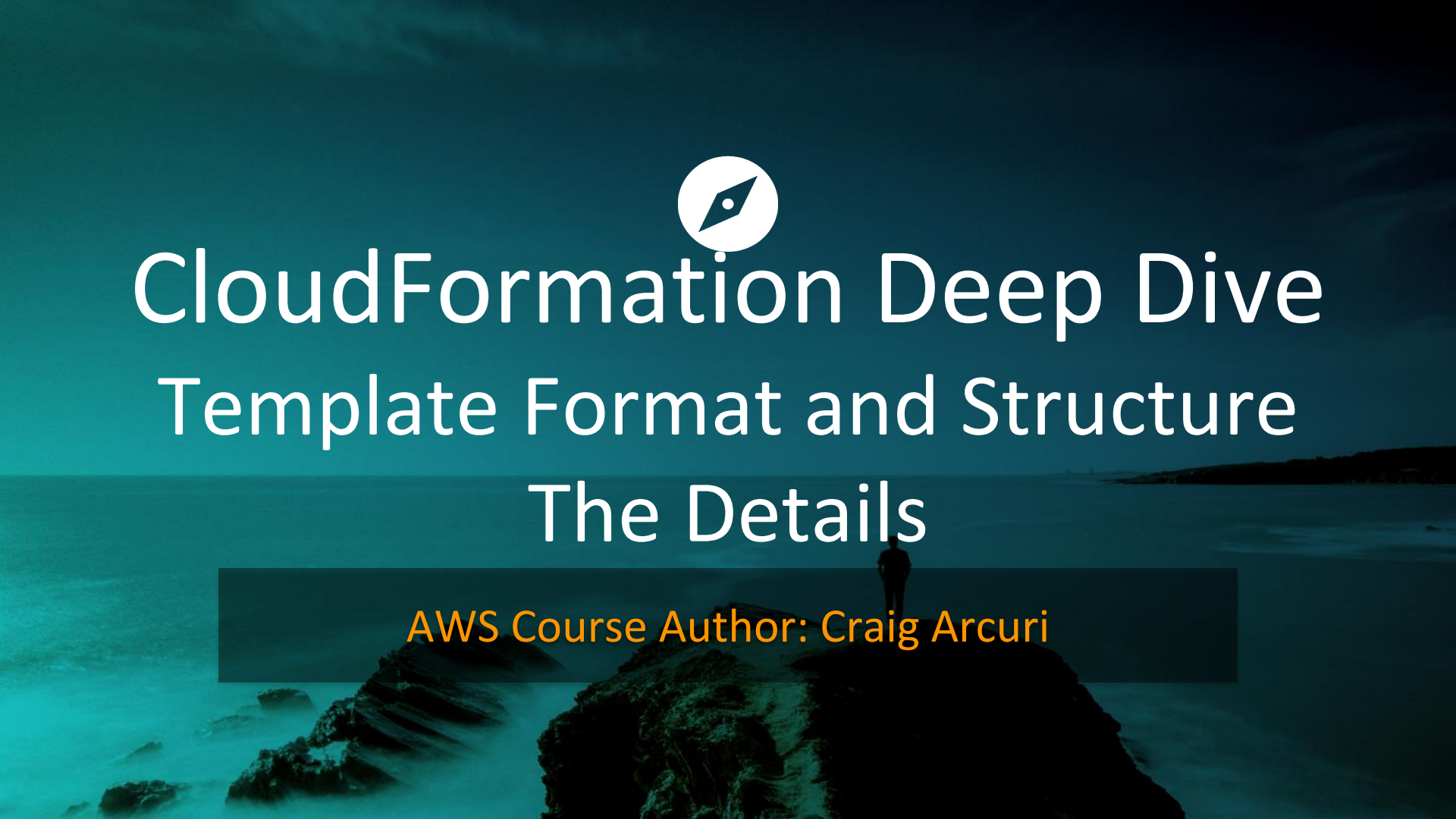
Creating Or Adding Stacks to Stack Sets

- Start small! Verify that adding stack instances to your initial stack set works before you more stack instances to your stack set.
- Choose the deployment options that are right for your use case:
    - Conservative deployment - set **Maximum Concurrent Accounts** to 1, and **Failure Tolerance** to 0. Set your lowest-impact region to be first in the **Region Order** list. Start with one region.
    - Fast Deployment - increase the values of **Maximum Concurrent Accounts** and **Failure Tolerance** as needed.
- Plan accordingly! Operations on stack sets depend on how many stack instances are involved, and can take significant time.

Updating Stacks in a Stack Set

- Remember, updating a stack set always touches all stack instances. Test the updated version of a template. Create a test stack set with the updated template, then add a few test accounts and deploy your template to the test stack set first.
- Updating a stack set with many stacks can take significant time. Plan your updates so you are not blocked from performing other operations on the stack set.


Thank you for watching!



CloudFormation Deep Dive

Template Format and Structure

The Details

AWS Course Author: Craig Arcuri

AWSTemplateFormatVersion

- Identifies the capability of the template.
- The latest template version is 2010-09-09.
- The only valid value currently.
- The value for the template format version declaration must be a literal string.
- JSON: "AWSTemplateFormatVersion" : "2010-09-09“
- YAML: AWSTemplateFormatVersion: "2010-09-09"

Description

- Optional
- Use to describe your template, provide versioning.
- Example:

- "Description" : “Three tier architecture, client xyz, version 1.0."

Metadata

- Optional
- Include arbitrary JSON or YAML objects that provide details about the template.

Parameters

- Pass values into your template when you create a stack.
- Each parameter must contain a value when you create a stack.
- Bounce out to console – split screen

```
"Parameters" : {
 "InstanceTypeParameter" : {
  "Type" : "String",
  "Default" : "t2.micro",
  "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],
  "Description" : "Enter t2.micro, m1.small, or m1.large. Default is t2.micro."
 }
}
```

Mappings

- Matches a key to a corresponding set of named values.

```
Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: Value02
    Key03:
      Name: Value03
```

Example in Visual Code

Conditions

- Includes statements that define when a resource is created or when a property is defined.
- You might use conditions when you want to reuse a template that can create resources in different contexts.
- Example Dev environment vs Prod Environment
    - Give Dev reduced capabilities to save money
- To use conditions, must include statements in at least 3 different sections of a template: Parameters – the values you want to evaluate, Conditions (Intrinsic Functions), Resource and Outputs (Associate conditions with the resources or outputs that you want to conditionally create.)

Ref function

- When declaring a resource and you need to specify another template resource by name, you can use the ref function to refer to the other resource.

CloudFormation Deep Dive

Template Best Practices

AWS Course Author: Craig Arcuri

Best Practices for Creating Templates

- Use input parameters to pass in information whenever you create or update a stack.
- Use constraints to validate templates – You can add constraints to parameters such as MinValue, MaxValue. You can insure that parameters entered by the user are valid and will not cause runtime errors.
- Protect sensitive data – Use noecho for passwords and other sensitive input data.
- Use Cloudtrail! – Integrate CloudTrail with CloudFormation to maintain logs of actions performed within CloudFormation. Good resource for audit compliance.

Best Practices for Creating Templates

- Tag your resources. Tagging is a best practice for creating resources outside of CloudFormation as well as within. CloudFormation provides stack name tags but custom tags can be added as well.
- Use AWS::CloudFormation::Init to deploy software on EC2 instances - you can install and configure software applications on Amazon EC2 instances by using the cfn-init helper script.
- Use the latest helper scripts – include: yum install -y aws-cfn-bootstrap

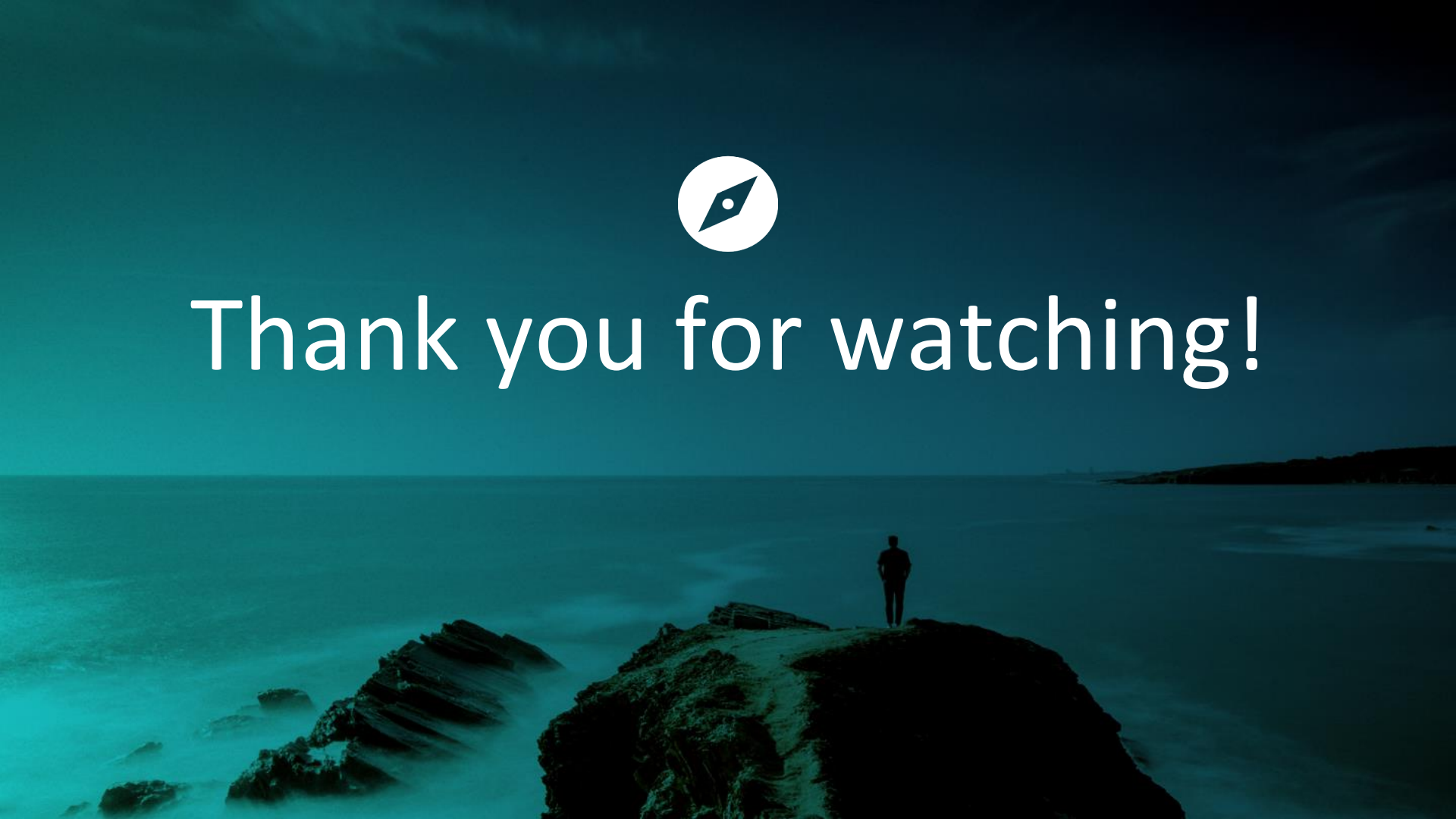
  in the user data section of your template before calling helper scripts.
- Validate your template before using it – in the console is one click (or automatically after parameters are input). CLI use – aws cloudformation validate-template.

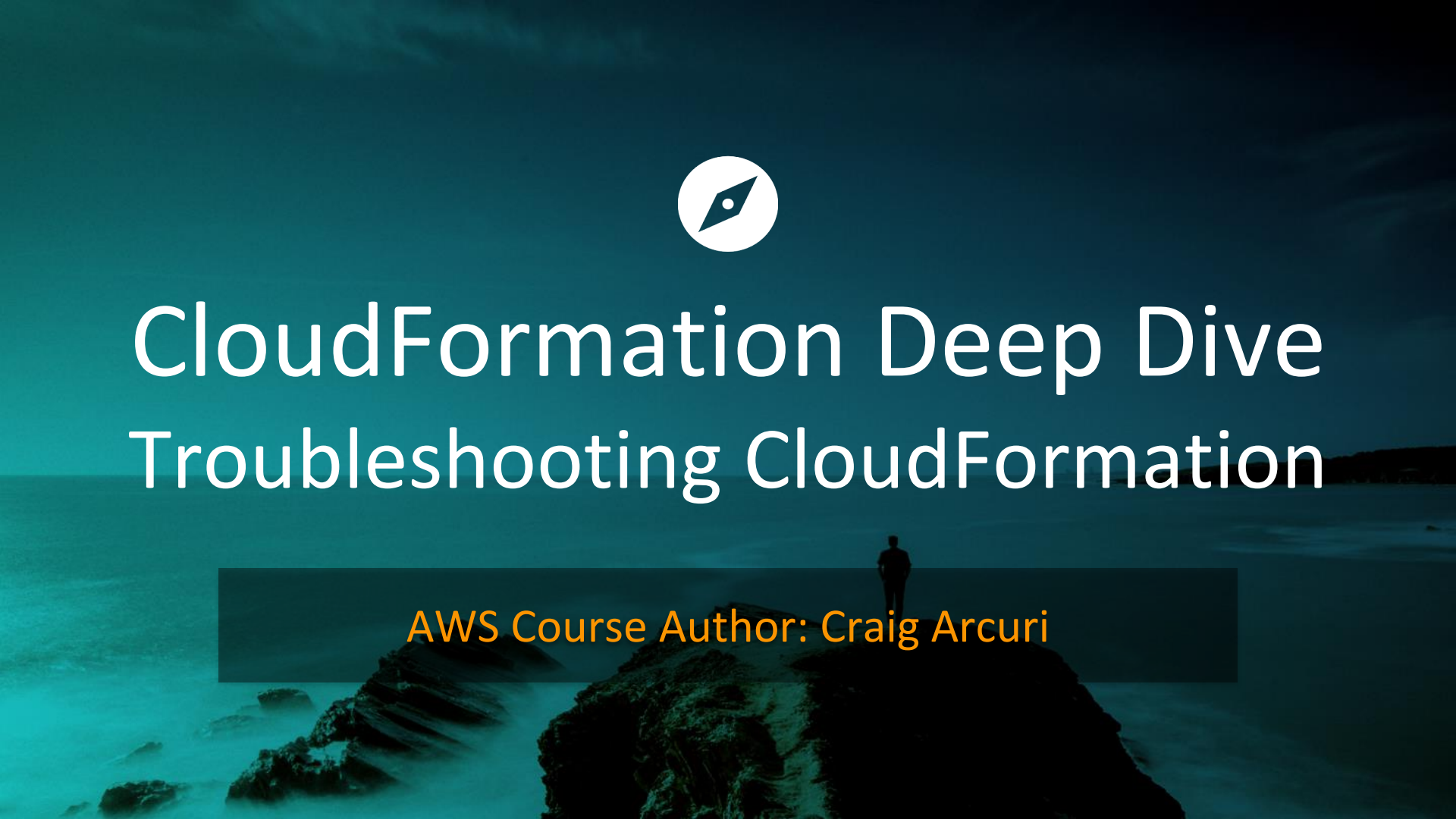
Best Practices for Creating Templates

- Version your templates!!! In the description section add a description and a version.
- Do not hard code AMI ID – use input parameters and mappings (could also use Lambda to get the ID).
- Use WaitCondition - a wait condition can be placed with in a template to make AWS CloudFormation pause the creation of the stack and wait for a signal before it continues to create the stack.
- Use DependsOn - specify that the creation of a specific resource DependsOn the creation of another resource.
- Cleanup after your stacks – When a stack is deleted, some artifacts such as snapshots and bucket logs may remain. Create scripts to cleanup.

Best Practices for Creating Templates

- Use Cloud-init to automate installation of software packages.
- Use helper scripts - CloudFormation provides Python helper scripts that you can use to install software and start services on EC2 instances in your stack.
- Use Intrinsic Functions to make your templates reusable and promote automation.
- Use nested stacks to simplify and keep the size of your templates manageable. The limit is 200 resources per template.


Thank you for watching!



CloudFormation Deep Dive

Troubleshooting CloudFormation

AWS Course Author: Craig Arcuri

Troubleshooting CloudFormation.

- CloudFormation is not a silver bullet. Yes, you can keep garbage out of your stacks with parameter constraints. But errors made in your architectural design and in your templates can be propagated to your stacks.
- You have created a stack with a Web Server within a VPC, with subnets, security groups, routing table, etc. But you are unable to SSH or RDP into the Web Server and there is no internet access. What's the problem? Have you created an IGW? Attached it to your VPC? Created a route to your IGW? Mind your architecture and configuration. CloudFormation won't catch this.
- But if your stack fails to create, update, or delete you can view error messages related to your stack. Use the CloudFormation Management Console to view the status of your stack.

Troubleshooting CloudFormation

- In the console, the events tab will list of stack events while your stack is being created, updated, or deleted. This list will also include rollback information if an error occurred.
- Find the failure event and then view the status reason for that event. The status reason might contain an error message CloudFormation or from a particular service that can help you troubleshoot your problem.
- Events Tab - All events that are triggered by a given stack operation are assigned the same client request token, which you can use to track operations. Stack operations that are initiated from the console use the token format Console-StackOperation-ID.
- For EC2 issues view the cloud-init and cfn logs in the /var/log/ directory on the instance. These logs capture processes and command outputs while AWS CloudFormation is setting up your instance.
- You can also configure your CloudFormation template to published to Amazon CloudWatch, which displays logs in the AWS Management Console so you don't have to connect to your Amazon EC2 instance.

Stack Delete Fails

- Some resources must be empty before they can be deleted. Example: You must delete all objects in an S3 bucket or remove all instances in a security group before you can delete the bucket or security group.
- You must have the necessary permissions to delete the resources in a stack. You must have permissions to the services in the stack in addition to CloudFormation permissions.
- If a stack is in the DELETE_FAILED state because CloudFormation couldn't delete a resource, rerun the deletion with the RetainResources parameter and specify the resource that CloudFormation couldn't delete. This will retain the troublesome resource but will at least allow you to delete the rest of the stack. You can then go about manually deleting the retained resource.
- You cannot delete a stack with termination protection enabled. The deletion will fail and the status will remain unchanged.
- It is recommended that you do not delete nested stacks directly, but only delete them as part of deleting the root stack and all its resources.

Troubleshooting CloudFormation

- Use DependsOn attribute to resolve a dependency error.
- You may have to explicitly declare dependencies so that CloudFormation can create or delete resources in the correct order.
- ‘Error Parsing Parameter When Passing a List’ – When you pass in a list, add the escape character (\) before each comma. Example:

ParameterKey=CIDR,ParameterValue='10.10.0.0/16\,10.10.0.0/24

- Insufficient IAM Permissions – To work with stacks you need CloudFormation permissions and also permission to use each individual resource within the stack.
- Invalid Value or Unsupported Resource Property - Your stack can fail due to invalid input parameters, unsupported resource property names, or unsupported resource property values. When you specify an Amazon EC2 key pair or VPC ID, the resource must exist in your account and in the region.

Troubleshooting CloudFormation

- Limit Exceeded - Verify that you didn't reach a resource limit. The default number Amazon EC2 instances that you can launch is 20. Delete excess resources or request a limit increase.
- Nested stacks get stuck in update processes - A nested stack failed to roll back. Because of potential resource dependencies between nested stacks, CloudFormation doesn't start cleaning up nested stack resources until all nested stacks have been updated or have rolled back. A nested stack might fail to roll back because of changes that were made outside of CloudFormation (Configuration Drift).
- No Updates To Perform - CloudFormation won't recognize some template changes as an update, such as changes to a deletion or update policy. Add or modify a metadata attribute in this case.

Troubleshooting CloudFormation

- Limit Exceeded - Verify that you didn't reach a resource limit. The default number Amazon EC2 instances that you can launch is 20. Delete excess resources or request a limit increase.
- Nested stacks get stuck in update processes - A nested stack failed to roll back. Because of potential resource dependencies between nested stacks, CloudFormation doesn't start cleaning up nested stack resources until all nested stacks have been updated or have rolled back. A nested stack might fail to roll back because of changes that were made outside of CloudFormation (Configuration Drift).
- No Updates To Perform - CloudFormation won't recognize some template changes as an update, such as changes to a deletion or update policy. Add or modify a metadata attribute in this case.

Troubleshooting CloudFormation

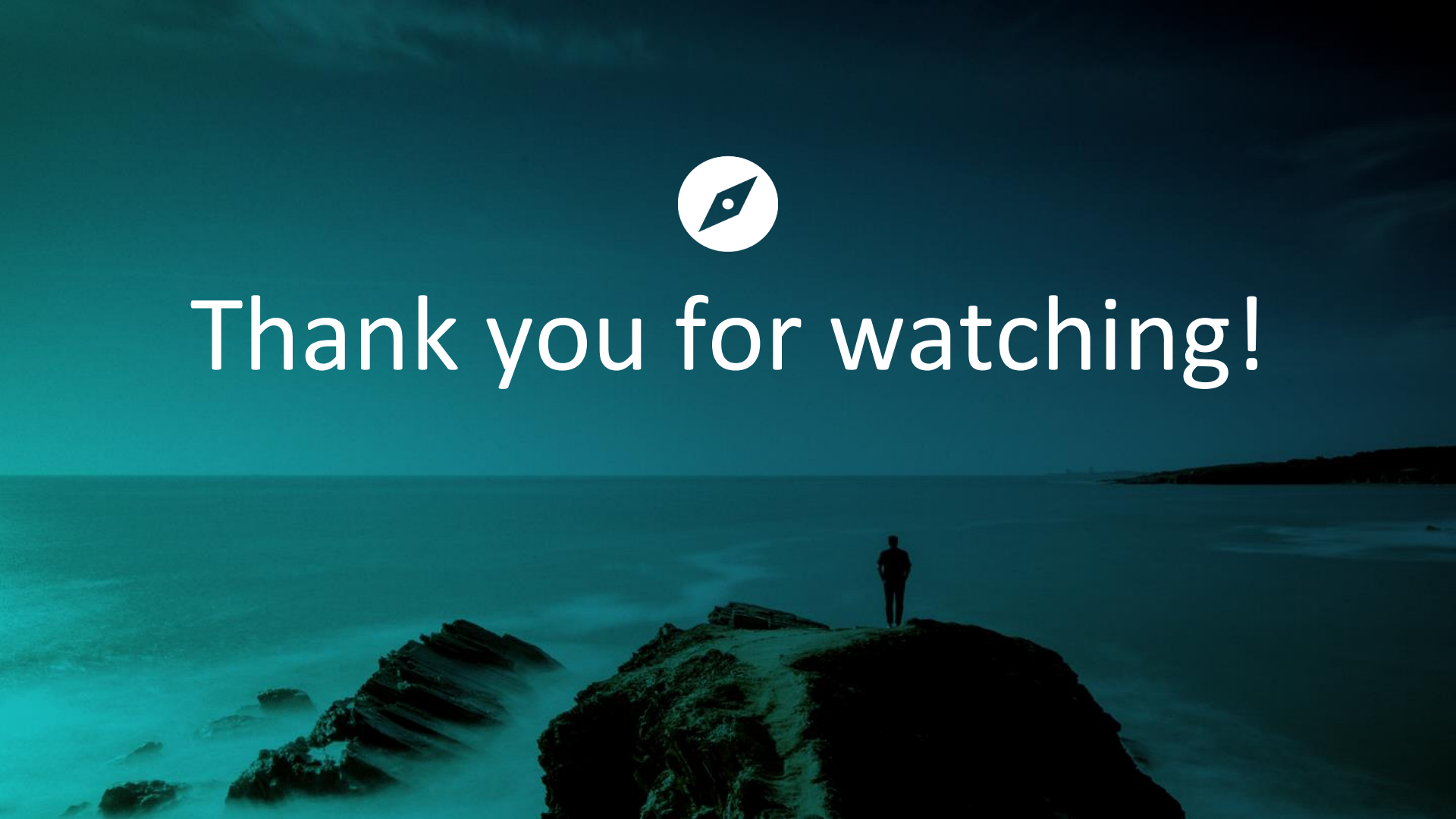
- Resource Failed to Stabilize During a Stack Operation - A resource did not respond because the operation exceeded the CloudFormation timeout period or an AWS service was interrupted. The timeout period depends on the resource and credentials that you use. Specify a service role to extend the timeout period when you perform the stack operation.
- Security Group does not exist in the VPC – Verify it's existence in the VPC. If the security group exists, ensure that you specify the security group ID and not the security group name.
- Update Rollback Failed - A dependent resource cannot return to its original state, causing the rollback to fail. Attempting to rollback to a resource that was deleted outside of CloudFormation (Configuration Drift). Manually fix the error and continue the rollback.

Update Rollback Failures

- Failed to receive the required number of signals – Use signal-resource command to manually send the required number of successful signals to the resource that is waiting for them, and then continue rolling back the update.
- Changes to a resource were made outside of CloudFormation - Manually sync resources so that they match the original stack's template, and then continue rolling back the update. AWS has announced Configuration Drift Detection coming in 2018.
- Insufficient Permissions – Verify sufficient IAM permissions to modify resources, and then continue the update rollback.
- Invalid security token - No change is required. Continue rolling back the update, which refreshes the credentials.
- Resource did not stabilize - A resource did not respond because the operation might have exceeded the CloudFormation timeout. No change is required. After the resource operation is complete continue rolling back the update.

Wait Condition Didn't Receive the Required Number of Signals from an Amazon EC2 Instance

- Ensure that the AMI you're using has the CloudFormation helper scripts installed.
- Verify that cfn-signal was successfully run on the instance. /var/log/cloud-init.log and /var/log/cfn-init.log can be used to debug. You can also publish the logs to CloudWatch.
- Verify that the instance has a connection to the Internet (Check IGW, routes, NAT if the instance is in a private subnet).


Thank you for watching!

CloudFormation Deep Dive

Update Policy

AWS Course Author: Craig Arcuri

What do Update Policies Do?

- Provide a structured way of updating instances in Auto Scaling groups with minimal downtime.
- Need to be able to update Auto Scaling Launch Configurations without causing downtime and updating all instances in the group.
- Having issues? How do you rollback?
- Can do all of this using Update Policies.
- Auto Scaling Group is a double edged sword. You can make changes to a group of EC2 instances, but the downside is that you can adversely affect a group of instances with one ill-advised changed.

Update Policy Attribute

- CloudFormation supports the update policy attribute.
- This is a resource attribute, and can be added to a resource in the stack.
- UpdatePolicy can be associated with AWS::AutoScaling::AutoScalingGroup resource to handle updates to that resources and the instances within the ASG.
- Update Policies can describe how the ASG and/or the instances within it are updated using 3 options:
- AutoScalingReplacingUpdate
- AutoScalingRollingUpdate
- AutoScalingScheduledAction

UpdatePolicy Options

- **AutoScalingReplacingUpdate** and **AutoScalingRollingUpdate** apply when changes are made to:
- The Launch Configuration
- The ASGs VPCZoneIdentifier property (subnets)
- When updating an ASG that has instances that don't match the Launch Configuration.
- **AutoScalingScheduledAction** applies when updating a stack that includes an ASG with an associated scheduled action.

AutoScalingReplacingUpdate Policy

- Specifies whether to replace an instance within an ASG or the whole ASG and its instances.
- WillReplace property (Boolean) , if true, the whole ASG and it's instances will be replaced.
- The old ASG is kept until the update is complete to enable rollback if an issue arises.
- YAML:

```
UpdatePolicy:
  AutoScalingReplacingUpdate:
    WillReplace: Boolean
```

AutoScalingReplacingUpdate With Creation Policy

- If WillReplace is set to true, you should specify how many instances need to signal success (with a Creation Policy) for the update to succeed.
- YAML:

```
UpdatePolicy:
  AutoScalingReplacingUpdate:
    WillReplace: Boolean
```

- If the requisite number of success signals are not received in the time specified by the timeout attribute, rollback is initiated.
- Can use “MinSuccessfulInstances” parameter to specify a percentage of instances that must return a success signal.
- Example: MinSuccessfulInstancesPercent : 50
- Note: Creation has priority over update if you have both in a policy.

AutoScalingRollingUpdate Policy

- Controls how many instances in an ASG are updated at the same time. This is done in batches.
- Also controls how many instances should be running during an update.
- Any issues during an update and you can automatically rollback to the previous version.
- Rolling updates enable you to update parts of your ASG while having a minimum amount of instances running. This can be used to eliminate downtime.
- Example: If you have an ASG with 6 instances, you can perform rolling update on 2 instances at a time and always insure that you have 4 instances available at all times.

AutoScalingRollingUpdate Syntax

JSON:

```
"UpdatePolicy" : {
  "AutoScalingRollingUpdate" : {
    "MaxBatchSize" : Integer,                ← max # of instance updated at once
    "MinInstancesInService" : Integer,
    "MinSuccessfulInstancesPercent" : Integer ←# instances needing to signal success
    "PauseTime" : String,
    "SuspendProcesses" : [ List of processes ],
    "WaitOnResourceSignals" : Boolean
  }
}
```

- PauseTime – amount of time CloudFormation pauses after making changes to a batch of instance. This allows time for bootstrapping.

AutoScalingRollingUpdate Syntax

```
YAML:
UpdatePolicy:
  AutoScalingRollingUpdate:
    MaxBatchSize: Integer
    MinInstancesInService: Integer
    MinSuccessfulInstancesPercent: Integer
    PauseTime: String
    SuspendProcesses:
      - List of processes
    WaitOnResourceSignals: Boolean
```

- SuspendProcesses – List of ASG processes to suspend during a stack update. Required for rolling updates and scheduled actions enabled. (Usese cfn-signal helper script).
- WaitOnResourceSignals – Force the ASG to wait on signals from individual instances before continuing. Used with PauseTime property. Need to receive a signal within the PauseTime period.

Benefits of Rolling Updates

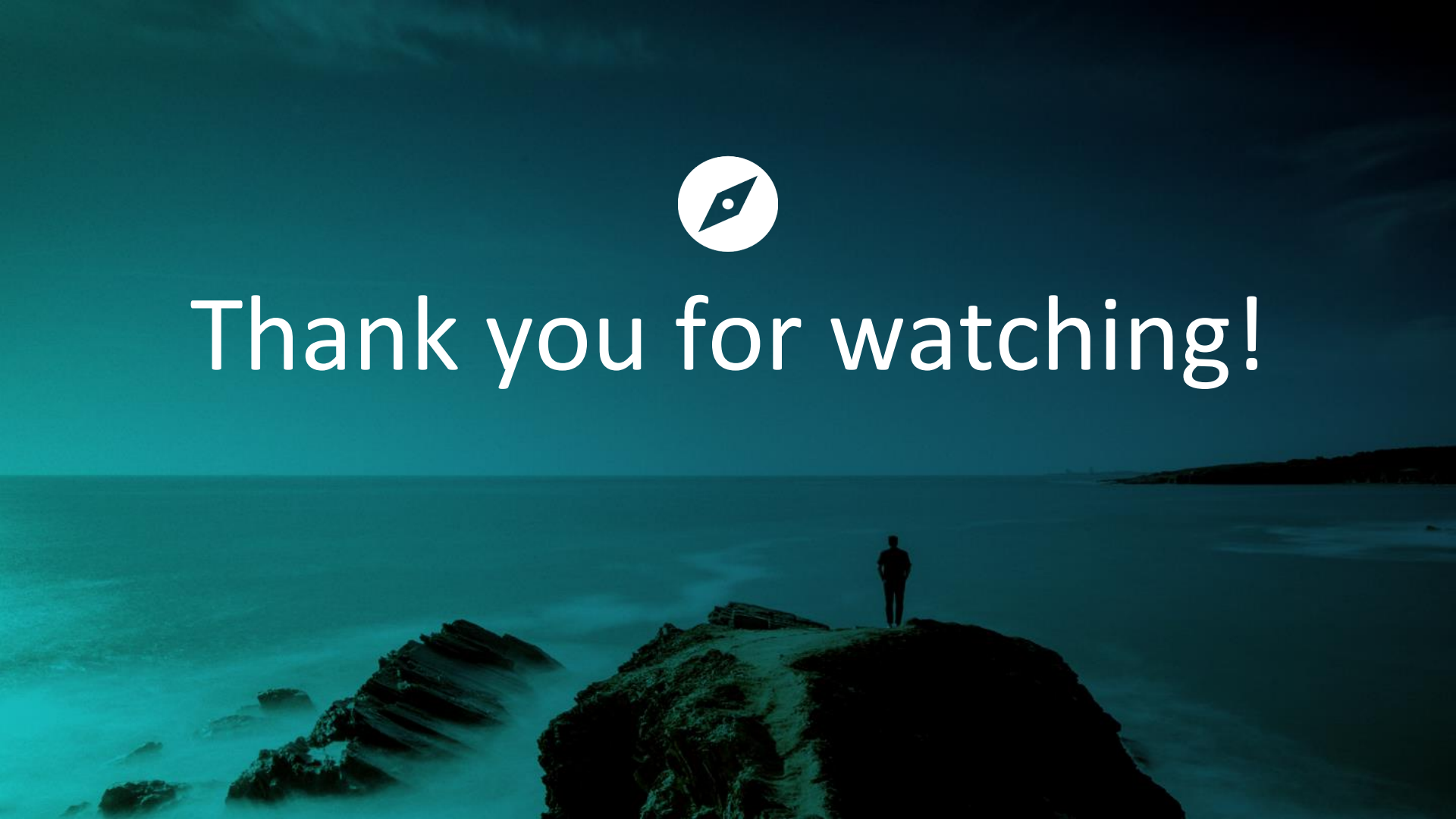
- Updating instance metadata ad using cfn-init and cfn-hup can cause downtime by updating instances at the same time.
- Inconsistencies can be caused by updating instance at different times.
- Rolling updates can solve these problems by updating instances one batch at a time within one update.
- Force a rolling update by changing the logical ID of the Launch Configuration resource, and then update the stack and any references to the original ID.
- This will cause the deletion of old instances and the creation of new ones. Back up your data!

AutoScalingScheduledAction Policy

- Can be used when you have predictable load patterns.
- Can use to handle updates for MinSize, MaxSize, and DesiredCapacity.
- Scheduled actions can change group size properties of an ASG. CloudFormation sets the group size property values back to the original values. This can be a problem.
- AutoScalingScheduledAction can be used to prevent changes by CloudFormation during updates. You can still update the values, but you have to explicitly modify the ASG group size properties.
- Syntax (YAML):

```
UpdatePolicy:
  AutoScalingScheduledAction:
    IgnoreUnmodifiedGroupSizeProperties: Boolean
```

- Specifies whether CloudFormation ignores differences in group size properties between the current ASG and the ASG defined in the template during an update.


Thank you for watching!

CloudFormation Deep Dive

Updating Stacks - Part 2

AWS Course Author: Craig Arcuri

Cancelling Updates (splitscreen)

- After a stack update has begun, you can cancel the stack update if the stack is still in UPDATE_IN_PROGRESS state.
- You can’t cancel an update after it has finished. But you can update back to the previous settings (make a copy of your template, then change the copy for updates. Don’t change the template in place. Follow good source control practice).
- Cancelling the stack update will force a rollback to the previous state of the stack.

Stack Policies

- All update actions are allowed on all resources by default.
- By default anyone with stack update permissions can update all of the resources in the stack.
- Stack policy can be used to protect stack resource from unintentional/mistaken update or delete.
- Stack policy - JSON document that defines the update actions that can be performed on designated resources.
- With stack policy - all resources in the stack are protected by default.
- Set an explicit ALLOW on a resource to allow an update to it.
- Only one stack policy per stack.

Stack Policies - continued

- A stack policy applies to all users who attempt to update the stack.
- A stack policy applies only during stack updates, it is not a replacement for IAM.

Stack Policies – example (splitscreen)

```
{
 "Statement" : [
  {
   "Effect" : "Allow",
   "Action" : "Update:*",
   "Principal": "*",
   "Resource" : "*"
  },
  {
   "Effect" : "Deny",
   "Action" : "Update:*",
   "Principal": "*",
   "Resource" : "LogicalResourceId/ProductionDatabase"
  }
 ]
}
```

Stack Policies – example (splitscreen)

- All resources are protected by default.
- Allow - Allows all actions on all resources.
- Explicit DENY overrides the allow. Prevents all update actions on the resource.
- The Principal element is required but only accepts (*).

Defining a Stack Policy

- When you set a stack policy on your stack, any update not explicitly allowed is denied by default.
- Five Elements in a Stack Policy: Effect, Action, Principal, Resource, and Condition.

Defining a Stack Policy

```
{
 "Statement" : [
  {
   "Effect" : "Deny_or_Allow",
   "Action" : "update_actions",
   "Principal" : "*",
   "Resource" : "LogicalResourceId/resource_logical_ID",
   "Condition" : {
    "StringEquals_or_StringLike" : {
     "ResourceType" : [resource_type, ...]
    }
   }
  }
 ]
}
```

Stack Policy Elements (splitscreen,setting a stackpolicy)

- Determines whether the actions that you specify are denied or allowed on the specified resource(s).
- Deny statement overrides an Allow.
- Specifies the update actions that are denied or allowed: Update:Modify, Update:Replace, Update:Delete, Update:*.
- Principal - specifies the entity that the policy applies to.
- Resource - Specifies the logical IDs of the resources that the policy applies to.
- Conditions – Specifies the resource type that the policy applies to.

CloudFormation Deep Dive

Updating Stacks

AWS Course Author: Craig Arcuri

CloudFormation Stack Updates

- Need to change a stack's settings or resources? Update the stack instead of deleting it and creating a new stack.
- Can submit new parameters or an updated template.
- CloudFormation compares the changes and updates only the changed resources.
- Update Methods:

  Direct Update – Used to quickly deploy updates

  Change Sets – Use to preview changes (no unintentional changes)
- Change sets allow you to preview changes and insure that no unexpected changes occur.

Update Behaviors

- CloudFormation updates resources based on differences between what you submit and the stack's current template.
- Resources that aren’t updated run without disruption during the update process.
- CloudFormation uses the following behaviors for resources being updated:
- **Update with no interruption**: Updates resource without disrupting operation of that resource and without changing the resource's physical ID.
- **Update with some interruption**: Updates the resource with some interruption and retains the physical ID.
- **Replacement**: Recreates the resource during an update, which also generates a new physical ID.

Update Behaviors - continued

- The update method used depends on which property you update for a given resource type. It's good to know a resources update behavior before performing an update (check here: AWS Resource Types Reference).
- The update behavior of a resource can dictate when to modify resources to reduce the impact of these changes on your application.

Modifying a Stack Template (splitscreen)

- To modify resources and properties of a Stack, you can modify the Stack's template.
- Use the template for the existing stack, and make your updates to that template.
- Get a copy of your stack template from your source control or from CloudFormation.
- Can update a template from the console or the CLI (another lesson).

Directly Updating Stacks (splitscreen)

- Submit a template or input parameters that specify updates to the resources in the stack.
- Updates deployed immediately.
- To use an existing template, make a copy and store locally or in S3 bucket.

Monitor Progress (splitscreen)

- Monitor the progress of a stack update by viewing the stack's events.
- Update process starts with anUPDATE_IN_PROGRESS event for the stack.
- After CloudFormation has successfully updated the stack you will see:
  UPDATE_COMPLETE
- If an update of a resource fails, you will see: UPDATE_FAILED
- If the update fails, CloudFormation rolls back any resources that it has updated during the upgrade to their configurations before the update.

CloudFormation Deep Dive Working With Stack Sets

AWS Course Author:
Craig Arcuri

Introduction to StackSets

- StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation.
- Using an administrator account, you define and manage an AWS CloudFormation template, and use the template as the basis for provisioning stacks into selected target accounts across specified regions.
- All the resources included in each stack are defined by the stack set's AWS CloudFormation template.

Introduction to StackSets

- There are pre-requisites to creating Stacksets
- Determine which AWS account is the *administrator account*. Stack sets are created in this account.
- A target account is the account in which you create individual stacks that belong to a stack set.
- Must have service roles configured that create a trust relationship between the two accounts.

Pre-requisite Configuration

- create an IAM role: **AWSCloudFormationStackSetAdministrationRole**
- https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/AWSCloudFormationStackSetAdministrationRole.yml

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/AWSCloudFormationStackSetExecutionRole"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Pre-requisite Configuration

- Target Account:

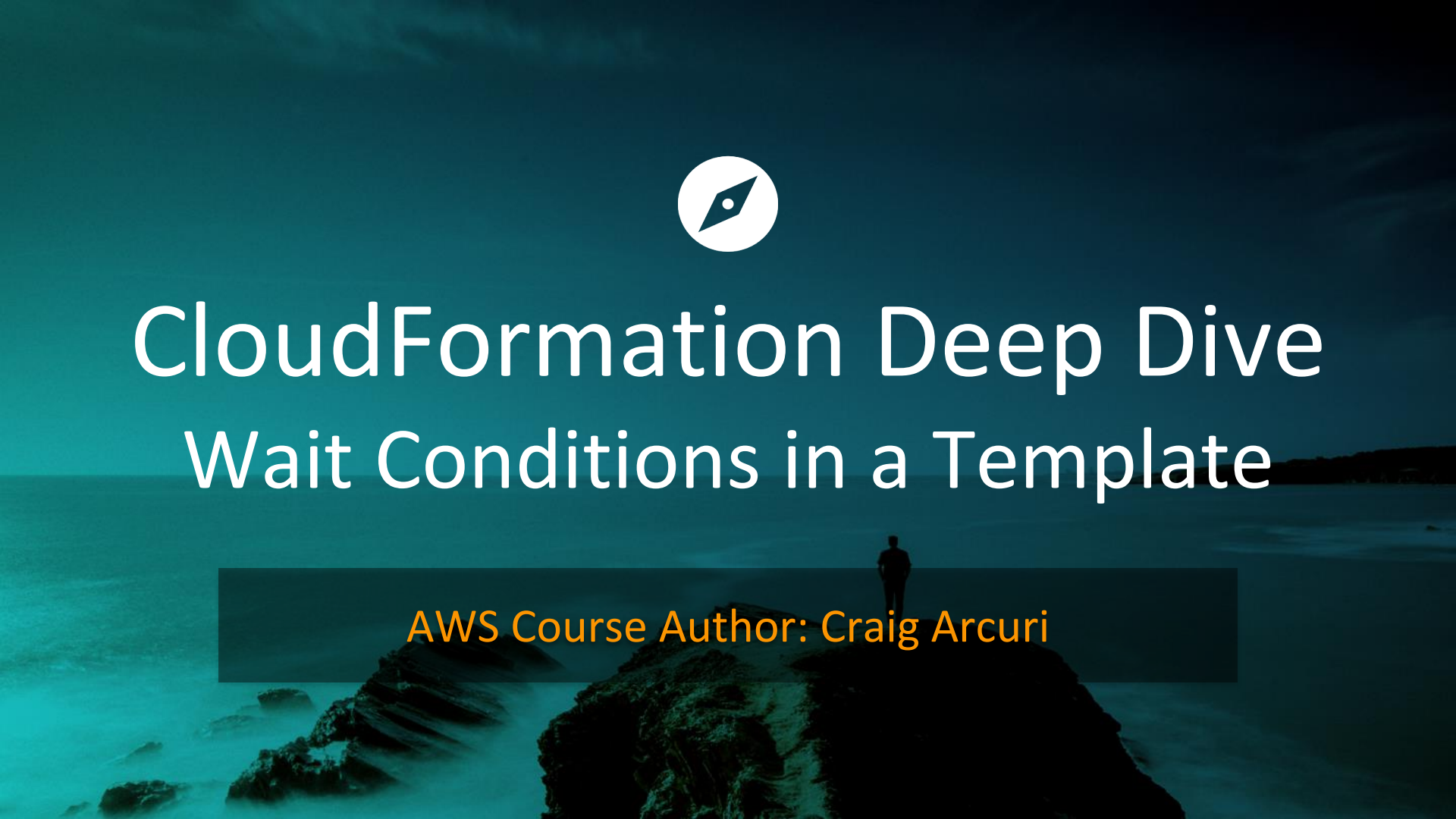
create a service role named **AWSCloudFormationStackSetExecutionRole**

https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/AWSCloudFormationStackSetExecutionRole.yml

Creates a policy:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "*",
         "Resource": "*"
      }
   ]
}
```


Thank you for watching!



CloudFormation Deep Dive

Wait Conditions in a Template

AWS Course Author: Craig Arcuri

Wait Conditions in Templates

- Use a WaitCondition to coordinate stack resource creation with configuration actions that are external to the stack creation.
- Use a WaitCondition to track the status of a configuration process.
- Note: AWS recommends using a CreationPolicy attribute instead of wait conditions for EC2 and Auto Scaling resources.
- You can start the creation of another resource after an application configuration is partially complete.
- You can send signals during an installation to track its progress.

Wait Condition Handle

- You can use the wait condition and wait condition handle to pause the creation of a stack and wait for a signal before it continues to create the stack.
- Helpful when the stack is dependent on resources create outside of the template that are needed for the stack.
- You might download and configure applications on an Amazon EC2 instance before considering the EC2 creation instance complete.

Wait Condition & Handle Operations

- CloudFormation creates a wait condition. Status is initially CREATE_IN_PROGRESS. Once it receives the requisite number of success signals it finishes creating the stack. Otherwise it sets the status to CREATE_FAILED and rolls back the stack.
- Timeout defines how long CloudFormation waits for the success signals.
- DependsOn attribute added to a wait condition determines that the wait condition is created only after the creation of a particular resource has completed (such as an EC2 or RDS DB instance).
- CloudFormation must receive the set number of success signals for a wait condition before setting that wait condition's status to CREATE_COMPLETE
- A wait condition requires a wait condition handle to set up a presigned URL as a signaling mechanism.

AWS::CloudFormation::WaitCondition

- JSON syntax:

```
{
  "Type" : "AWS::CloudFormation::WaitCondition",
  "Properties" : {
    "Count" : Integer,
    "Handle" : String,
    "Timeout" : String
  }
}
```

AWS::CloudFormation::WaitCondition

- YAML syntax:

```
Type: "AWS::CloudFormation::WaitCondition"
Properties:
  Count: Integer
  Handle: String
  Timeout: String
```

WaitCondition Properties

- Count - The number of success signals that CloudFormation must receive before it continues the stack creation process. If the wait condition does not receive the specified number of success signals before the Timeout period expires, the stack is rolled back.
- Handle - A reference to the wait condition handle used to signal this wait condition. Use ref to specify a WaitConditionHandle resource. Anytime you add a WaitCondition resource during a stack update, you must associate the wait condition with a new WaitConditionHandle resource. Reusing a wait condition handle may cause problems. The wait condition might evaluate old signals from a previous create or update stack command.

WaitCondition Properties

- Timeout – The length of time to wait for the number of signals specified by the count property. The maximum time that can be specified is 12 hours.

Return Values:

- Ref - The logical ID of a resources is provided to the Ref function which returns the resource name.
- Fn::GetAtt - returns a value for a specified attribute of this type.

WaitCondition Syntax

```
WebServerGroup:
  Type: "AWS::AutoScaling::AutoScalingGroup"
  Properties:
    AvailabilityZones:
      Fn::GetAZs: ""
    LaunchConfigurationName:
      Ref: "LaunchConfig"
    MinSize: "1"
    MaxSize: "5"
    DesiredCapacity:
      Ref: "WebServerCapacity"
    LoadBalancerNames:
      -
        Ref: "ElasticLoadBalancer"
```

WaitCondition Example

```
WaitHandle:
  Type: "AWS::CloudFormation::WaitConditionHandle“      (note: has no properties)
WaitCondition:
  Type: "AWS::CloudFormation::WaitCondition"
  DependsOn: "WebServerGroup"
  Properties:
    Handle:
      Ref: "WaitHandle"
    Timeout: "300"
    Count:
      Ref: "WebServerCapacity"
```

WaitCondition – Key Points

- WaitHandle has no properites but it is required.
- Using Ref with the handle resource logical name gets access to the pre-signed S3 url. Success or failure signals are sent by the url.
- Can also pass the url in to another resource, such as an EC2 instance so it can send success/failure signals.
- Send an HTTP request with the pre-signed url to send success signals (this should be a PUT request in a JSON format).

Wait Condition Use Cases

- Synchronize resource creation between different resources in the template.
- Waiting for external resources to be created

- Anti-patterns:
- Do not use for EC2 instances or Autoscaling groups. Use Creation Policies instead.

Using a Wait Condition in a Stack - 1

- Declare AWS::CloudFormation::WaitConditionHandle:

```
"myWaitHandle" : {
    "Type" : "AWS::CloudFormation::WaitConditionHandle",
    "Properties" : {
    }
}
```

Using a Wait Condition in a Stack - 2

- Declare AWS::CloudFormation::WaitCondition:

```
"myWaitCondition" : {
   "Type" : "AWS::CloudFormation::WaitCondition",
   "DependsOn" : "Ec2Instance",
   "Properties" : {
      "Handle" : { "Ref" : "myWaitHandle" },
      "Timeout" : "4500"
   }
}
```

Using a Wait Condition in a Stack - 3

- Getting the signaling pre-signed url:

```
"UserData" : {
  "Fn::Base64" : {
    "Fn::Join" : [ "", ["SignalURL=", { "Ref" : "myWaitHandle" } ] ]
  }
}
```

Using a Wait Condition in a Stack - 4

- Select a method for detecting when the stack enters the wait condition.
- If you create the stack with notifications enabled, AWS CloudFormation publishes a notification for every stack event to the specified topic.
- You can also monitor the stack's events using the Management Console, the CLI, or CloudFormation API.

Using a Wait Condition in a Stack - 5

- Use the presigned URL to signal success or failure (send an HTTP request message using the presigned URL).
- Curl command with JSON structure as a parameter:

```
{
  "Status" : "SUCCESS",
  "Reason" : "Configuration Complete",
  "UniqueId" : "ID1234",
  "Data" : "Application has completed configuration."
}
```