
AWS
re:Invent

DOP303

Choosing the right modern deployment strategy

Andrew Baird

Principal Solutions Architect

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

Continuous integration (CI) / continuous development (CD) – Refresher

Modern deployment tenets

Modern deployment considerations

Modern deployment options

Conclusion



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

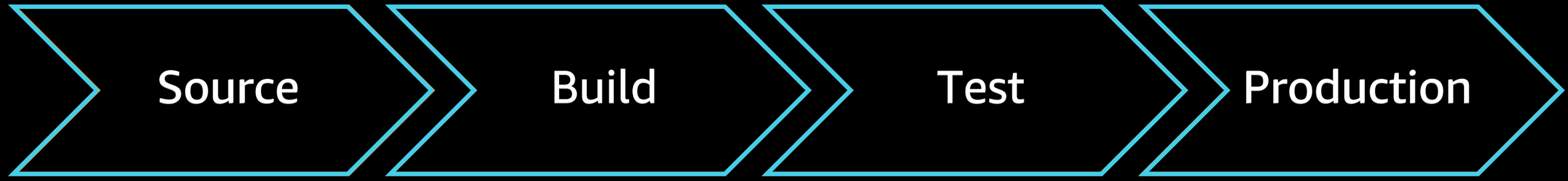
CI / CD refresher



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Release process stages



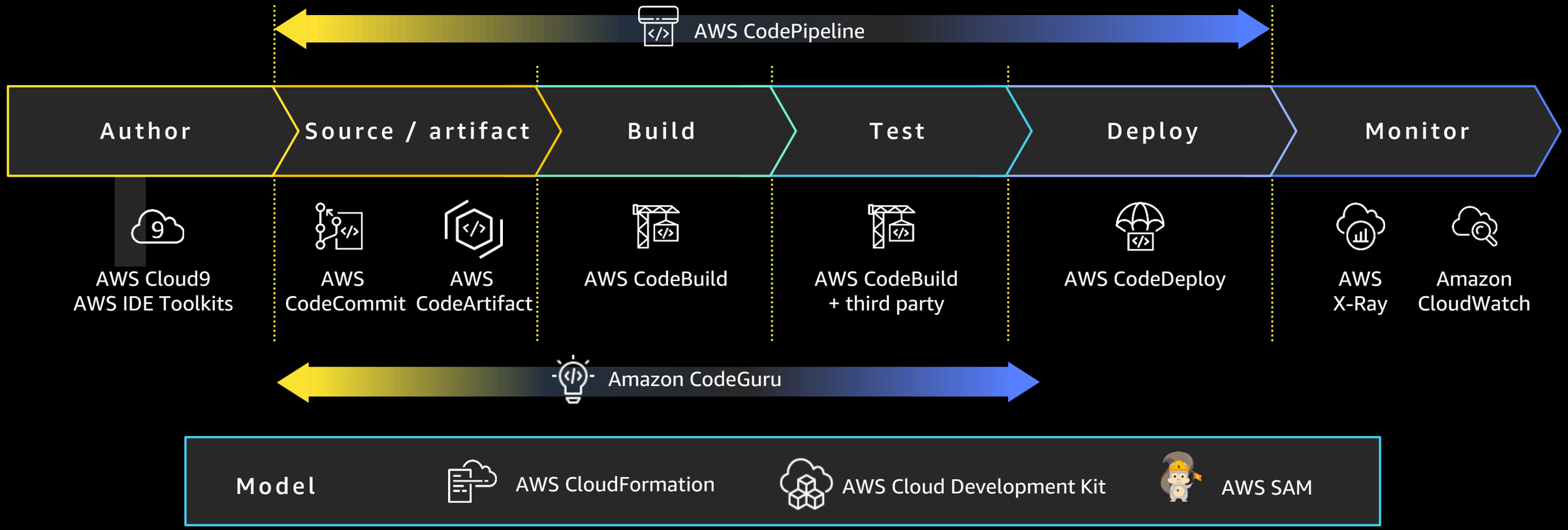
- Check-in source code such as .java files
- Peer review new code

- Compile code
- Unit tests
- Style checkers
- Create build artifacts

- Integration tests with other systems
- Load testing
- UI tests
- Security testing

- Deployment to production environments
- Monitor code in production to quickly detect errors

AWS Developer Tools for modern software delivery



AWS CodeDeploy

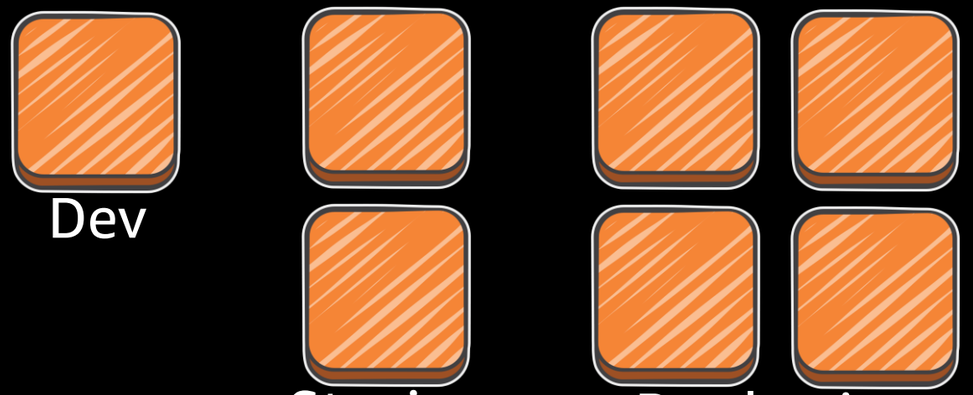
Application revisions



v1, v2, v3



Deployment groups



Scale from single instance to thousands

Multiple configurations for zero downtime deployment

Centralize deployment control and monitoring

Configurable deployment hooks

Deployment for servers, containers, and AWS Lambda

Modern deployment tenets



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

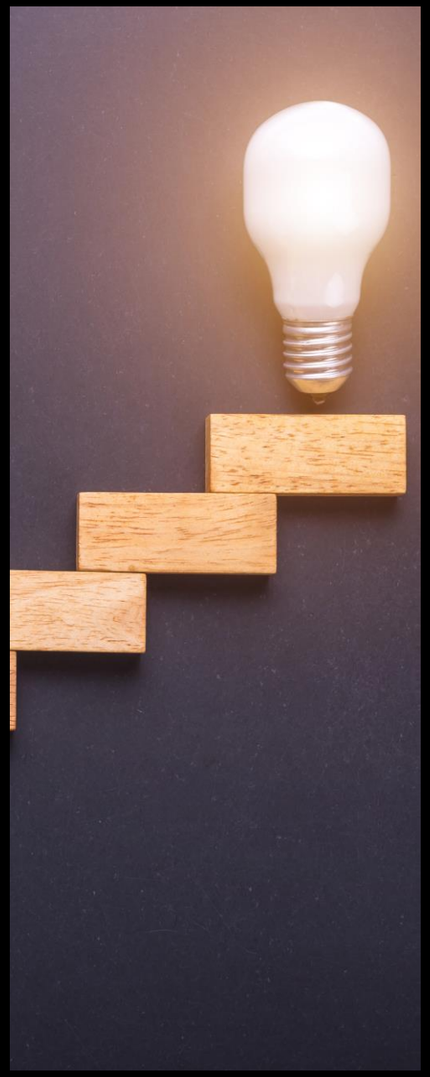
Modern deployment tenets



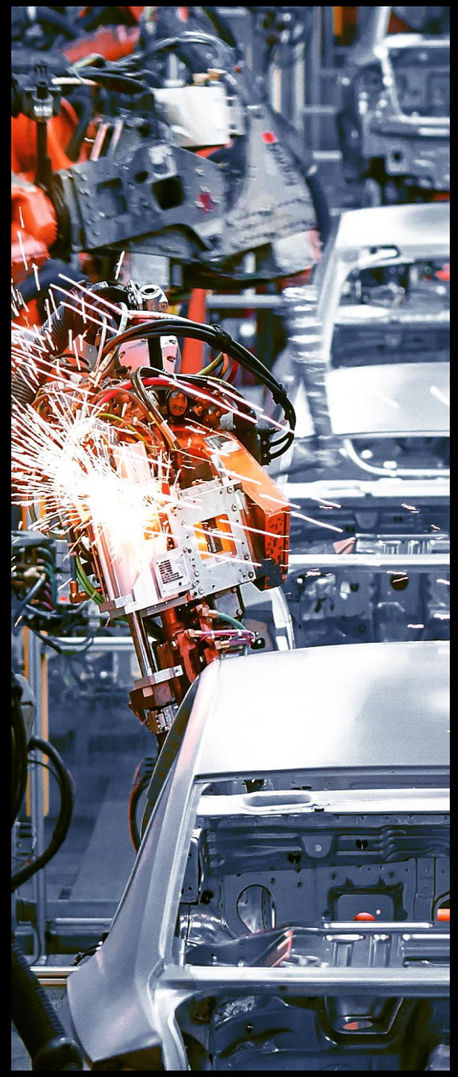
No business disruption



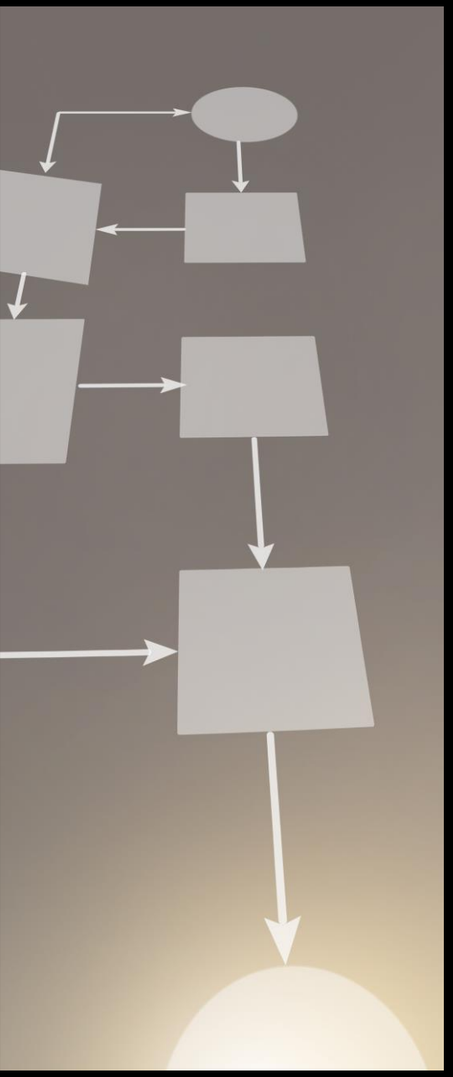
Iterative and frequent



Hardened versions



Automated



Auditable

Modern deployment considerations



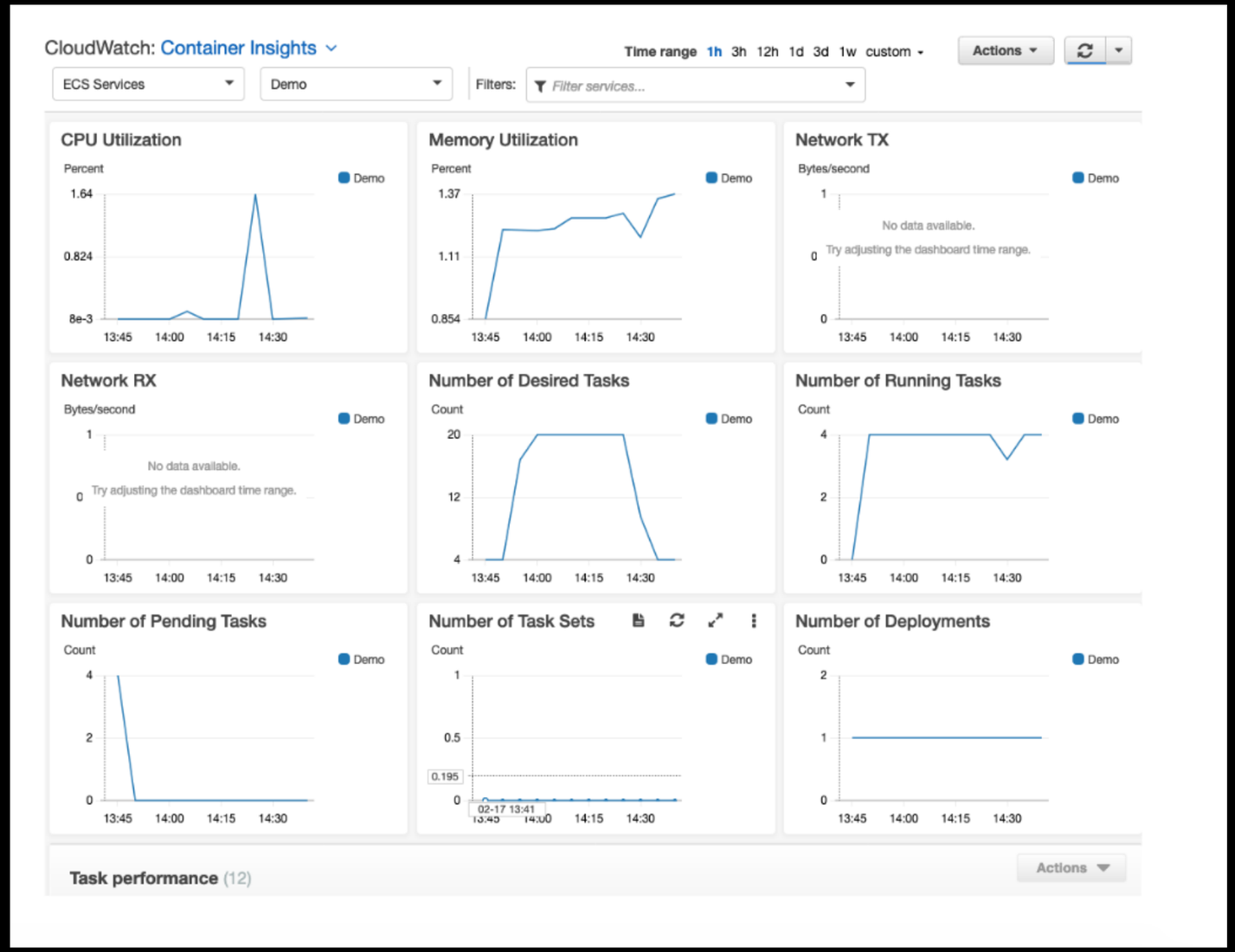
AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Metrics, tests, and alarms

MODERN DEPLOYMENT CONSIDERATIONS

- Deployments are only as trustworthy as your metrics, test suites, and alarms
- Leverage built-in Amazon CloudWatch metrics
- Custom application Amazon CloudWatch metrics are necessary
- Alarms prevent deployment, automate rollback



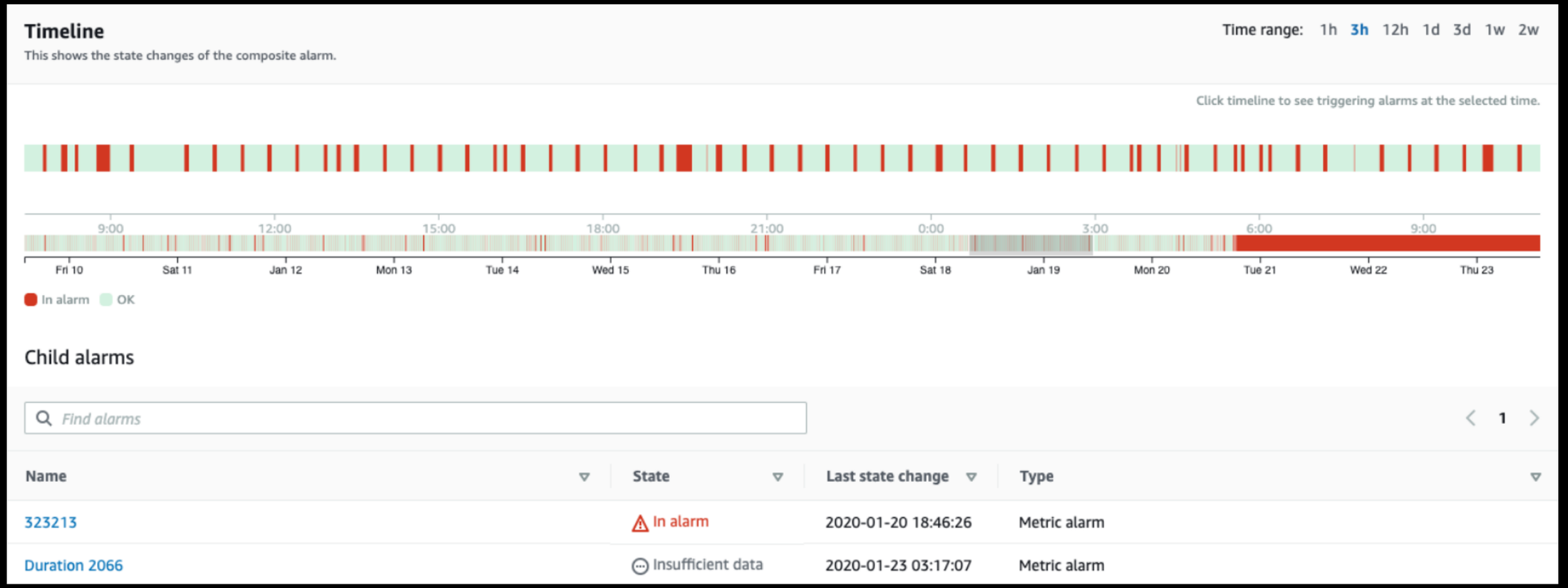
Metrics, tests, and alarms

MODERN DEPLOYMENT CONSIDERATIONS

Pro tip

Amazon CloudWatch composite alarms

```
(ALARM("CPUUtilizationTooHigh") OR
ALARM("DiskReadOpsTooHigh")) AND
OK("NetworkOutTooHigh")
```



Mutable vs. immutable infrastructure

MODERN DEPLOYMENT CONSIDERATIONS

Immutable infrastructure – After infrastructure is made active within an application environment, it is no longer permitted to change



Mutable infrastructure

Pro – Persisted state between deployments

Pro – Hot-fix velocity

Con – Rollback velocity

Con – Operational investigation complexity



Immutable infrastructure

Pro – Automation enforcement / consistency

Pro – Rollback velocity

Con – Hot-fix velocity

Con – Cost

Modern deployment options



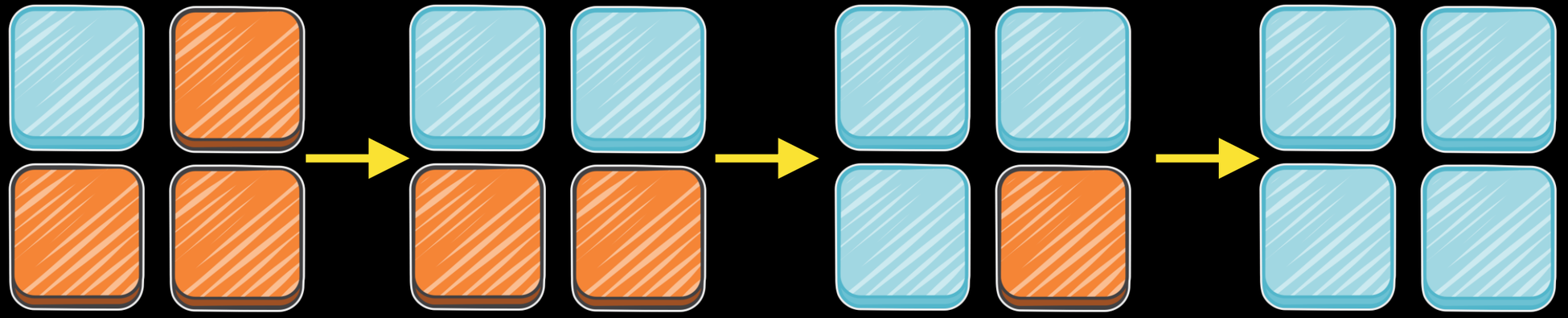
AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Rolling or linear deployment

MODERN DEPLOYMENT OPTIONS

An incrementally increasing percentage of requests is serviced by the newly deployed application version until 100% is reached



Pros

- Adds risk incrementally
- Limits concurrent changes
- Reuses infrastructure

Cons

- Deployment completion time
- Rollback completion time
- Heterogeneous environment

Implementing: Rolling or linear deployment

AMAZON ELASTIC COMPUTE CLOUD (AMAZON EC2)

AWS CodeDeploy

```
MyDeploymentConfig:
    Type: AWS::CodeDeploy::DeploymentConfig
    Properties:
      MinimumHealthyHosts:
        Type: FLEET_PERCENT
        Value: '90'
MyDeploymentGroup:
    Type: AWS::CodeDeploy::DeploymentGroup
    Properties:
      ApplicationName: MyApplication
      DeploymentConfigName: !Ref MyDeploymentConfig
      DeploymentGroupName: PROD-MyApplication
      DeploymentStyle:
        DeploymentOption: WITH_TRAFFIC_CONTROL
        DeploymentType: IN_PLACE
      LoadBalancerInfo:
        ElbInfoList:
          - Name: MyApplicationProdLoadBalancer
        TargetGroupInfoList:
          - Name: MyApplicationTargetGroup
```

AWS Auto Scaling

```
Type: AWS::AutoScaling::AutoScalingGroup
Properties:
  AutoScalingRollingUpdate:
    MaxBatchSize: 2
    MinInstancesInService: 10
    MinSuccessfulInstancesPercent: 100
    PauseTime: 300
    SuspendProcesses:
    - List of Processes
    WaitOnResrouceSignals: True
```

Implementing: Rolling or linear deployment

AMAZON ELASTIC CONTAINER SERVICE (AMAZON ECS) AND AWS LAMBDA

Amazon ECS

```
Type: AWS::ECS::Service
Properties:
  DeploymentConfiguration:
    MaximumPercent: 110
    MinimumHealthyPercent: 100
  DeploymentController:
    Type: ECS
```

AWS Lambda

```
Resources:
 MyLambdaFunction:
   Type: AWS::Serverless::Function
   Properties:
     Handler: index.handler
     Runtime: nodejs12.x
     CodeUri: s3://bucket/code.zip

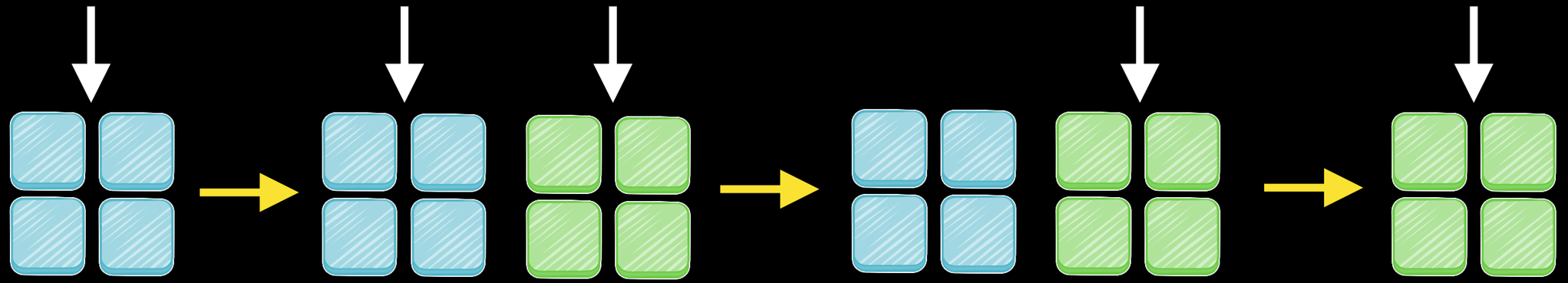
     AutoPublishAlias: live

     DeploymentPreference:
       Type: Linear10Percent3Minutes
       Alarms:
         - !Ref AliasErrorMetricGreaterThanZeroAlarm
         - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
       Hooks:
         PreTraffic: !Ref PreTrafficLambdaFunction
         PostTraffic: !Ref PostTrafficLambdaFunction
```

Blue / green deployment

MODERN DEPLOYMENT OPTIONS

Deploy net-new infrastructure (such as servers and containers) that contain the updated application version; shift requests from the old infrastructure to the new infrastructure incrementally; retain the old infrastructure for rollback purposes



Pros

- Environment promotion consistency
- Simple deployment / rollback mechanics

Cons

- Cost / capacity required
- Hot-fix rigidity
- Cold infrastructure performance

Implementing: Blue / green deployment

Option 1: Create a new environment (required for Amazon EC2)
or . . .

Amazon ECS

```
MyDeploymentGroup:
    Type: AWS::CodeDeploy::DeploymentGroup
    Properties:
      ApplicationName: MyEcsApplication
      DeploymentConfigName: CodeDeployDefault.ECSLinear10Percent1Minutes
      DeploymentGroupName: PROD-MyEcsApplication
      DeploymentStyle:
        DeploymentOption: WITH_TRAFFIC_CONTROL
        DeploymentType: BLUE_GREEN
      LoadBalancerInfo:
        ElbInfoList:
          - Name: MyApplicationProdLoadBalancer
        TargetGroupInfoList:
          - Name: MyApplicationTargetGroup
```

AWS Lambda

```
Resources:
 MyLambdaFunction:
   Type: AWS::Serverless::Function
   Properties:
     Handler: index.handler
     Runtime: nodejs12.x
     CodeUri: s3://bucket/code.zip

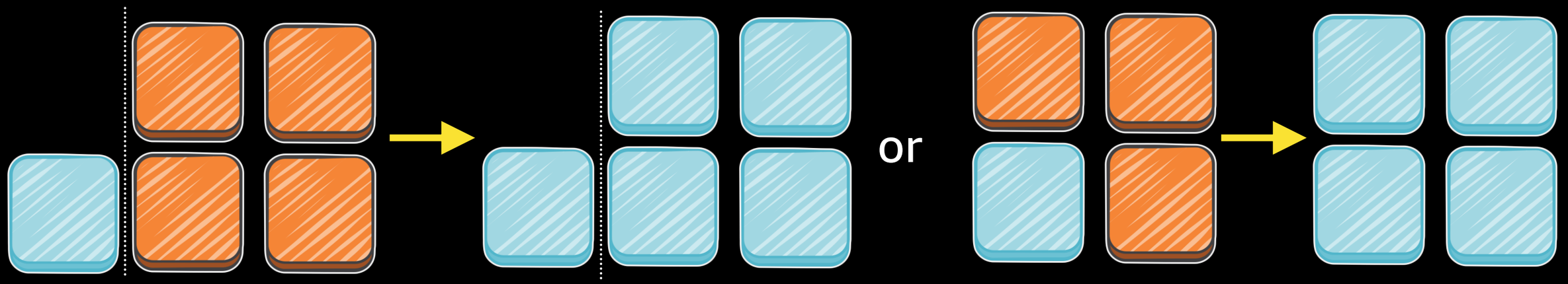
     AutoPublishAlias: live

     DeploymentPreference:
       Type: Linear10Percent3Minutes
       Alarms:
         - !Ref AliasErrorMetricGreaterThanZeroAlarm
         - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
       Hooks:
         PreTraffic: !Ref PreTrafficLambdaFunction
         PostTraffic: !Ref PostTrafficLambdaFunction
```

Canary or one-box deployment

MODERN DEPLOYMENT OPTIONS

First, deploy to the smallest unit of infrastructure possible; if successful, deploy to remaining units (all-at-once or rolling / linear)



Pros

- Minimizes risk
- Enables experimentation

Cons

- Multi-staged deployment / rollback
- Complexity

Implementing: Canary or one-box deployment

Option 1: Create a new environment (required for Amazon EC2)
or . . .

Amazon ECS (with AWS CodeDeploy)

```
MyDeploymentGroup:
    Type: AWS::CodeDeploy::DeploymentGroup
    Properties:
      ApplicationName: MyEcsApplication
      DeploymentConfigName: CodeDeployDefault.ECSCanary10Percent5Minutes
      DeploymentGroupName: PROD-MyEcsApplication
      DeploymentStyle:
        DeploymentOption: WITH_TRAFFIC_CONTROL
        DeploymentType: BLUE_GREEN
      LoadBalancerInfo:
        ElbInfoList:
          - Name: MyApplicationProdLoadBalancer
        TargetGroupInfoList:
          - Name: MyApplicationTargetGroup
```

AWS Lambda

```
Resources:
 MyLambdaFunction:
   Type: AWS::Serverless::Function
   Properties:
     Handler: index.handler
     Runtime: nodejs12.x
     CodeUri: s3://bucket/code.zip

     AutoPublishAlias: live

     DeploymentPreference:
       Type: Canary10Percent10Minutes
       Alarms:
         - !Ref AliasErrorMetricGreaterThanZeroAlarm
         - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
       Hooks:
         PreTraffic: !Ref PreTrafficLambdaFunction
         PostTraffic: !Ref PostTrafficLambdaFunction
```

Conclusion: Choosing the right deployment option



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Choosing the right deployment option

Step 1 – Metrics, tests, and alarms

Step 2 – Automate

Step 3 – Iterate



Additional thoughts

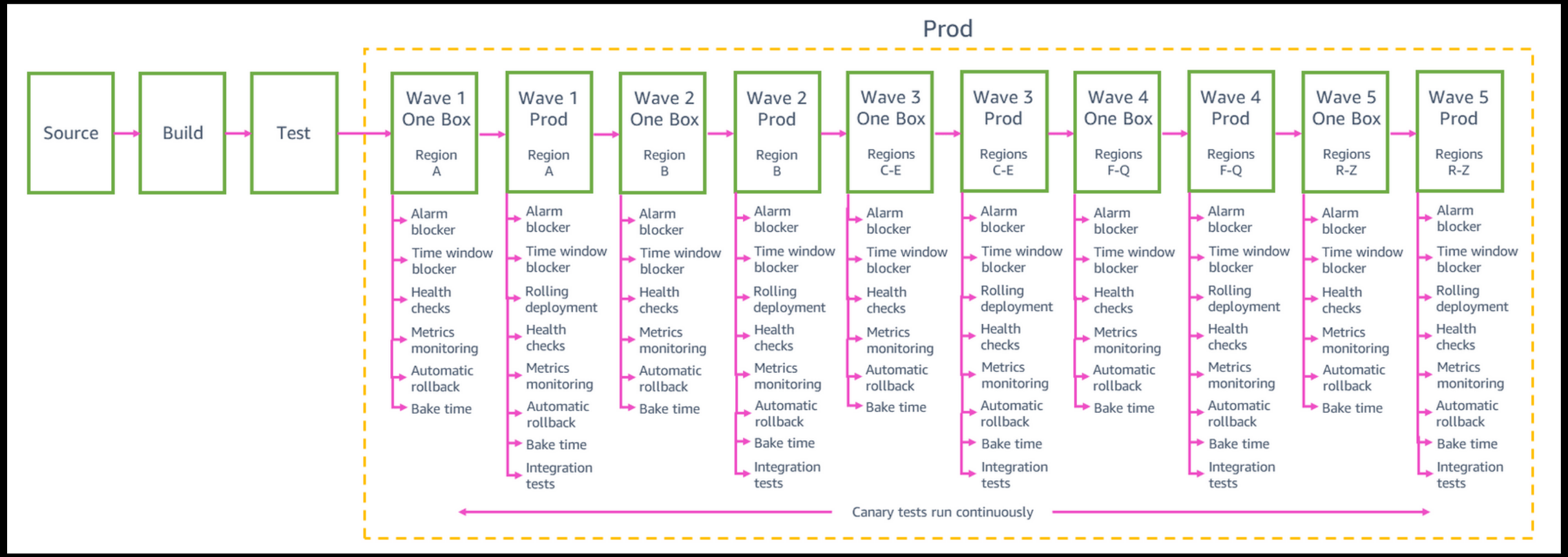
What deployment options are available to you?

- Compute service relevance (for example, no in-place deployment of AWS Lambda functions)
- Can your budget support the scale / capacity required for blue / green

Which downsides would be most impactful for your application?

- Dependency on state (warm local caches, DB connection pools, etc.)
- Need for frequent hot-fixing

Safe hands-off deployment at Amazon



https://aws.amazon.com/builders-library/automating-safe-hands-off-deployments/

Related session

BLD 207 Hands-off: Automating continuous delivery pipelines at Amazon

As a developer, manually deploying code to production can be a daunting series of tasks. Am I deploying the right changes? What metrics do I need to watch? Should I rollback? Each step could go wrong due to human error and cause teams to scramble. Amazon has taken an automated approach to continuous delivery that helps developers release code safely and quickly, with pipelines that enable us to focus on building solutions rather than managing deployments. Join Clare Liguori as she describes Amazon’s approach to continuous delivery and how it has improved her team’s ability to deliver changes to production.

Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.