AWS re:Invent

CON333

Best practices for CI/CD using AWS Fargate and Amazon ECS

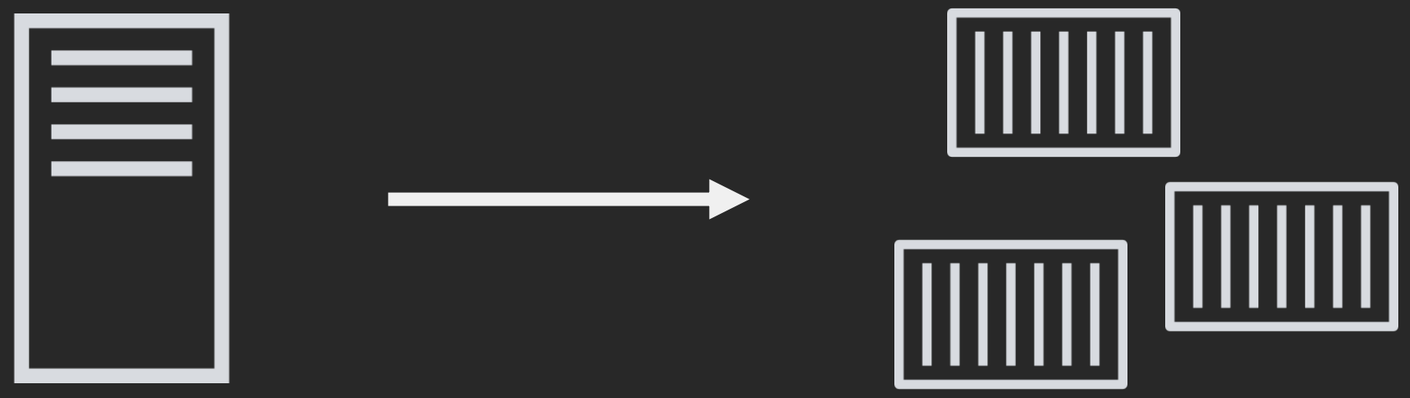
Clare Liguori
Principal Engineer
Amazon Web Services

Hsing-Hui Hsu
Software Engineer
Amazon Web Services

AWS re:Invent

aws

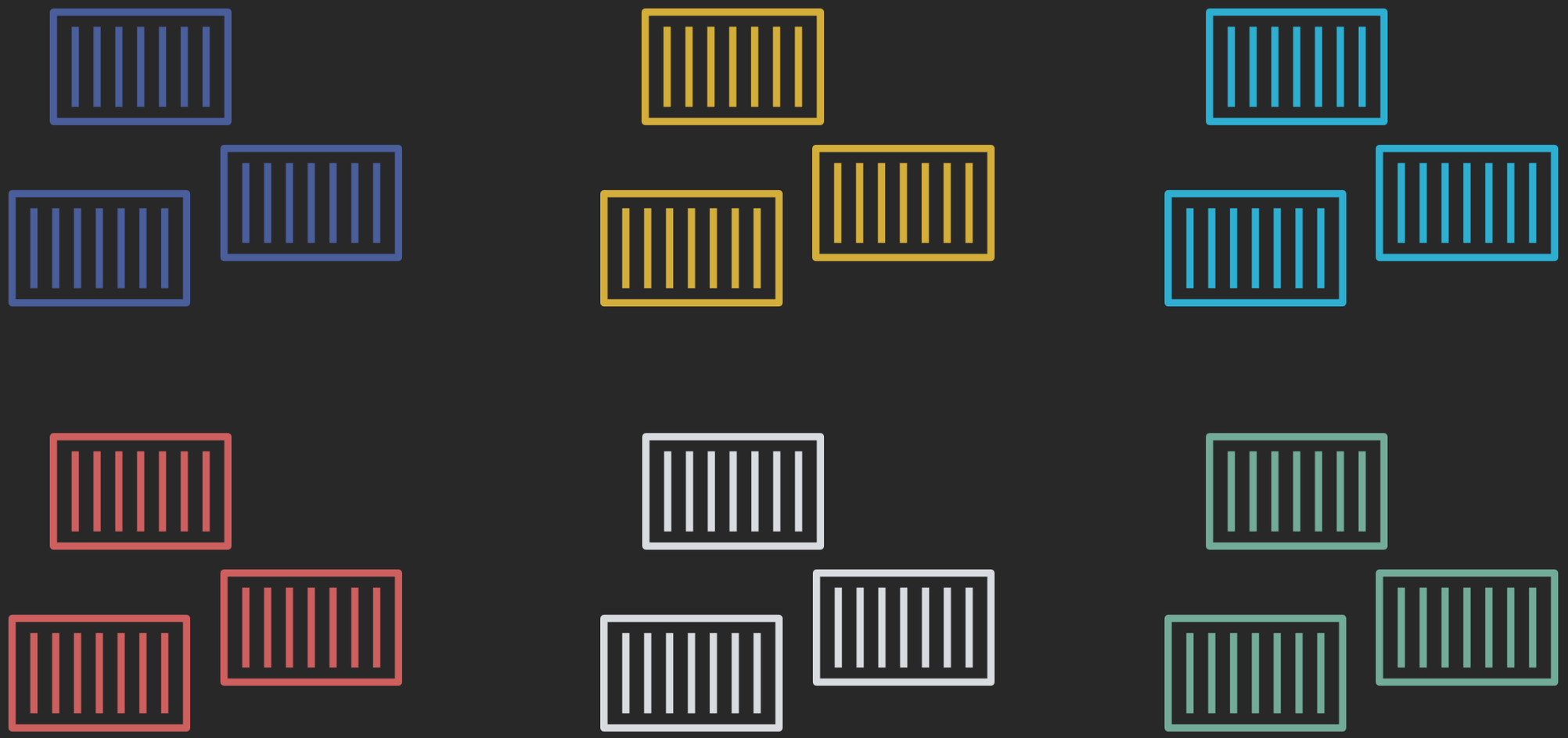
Containers and CI/CD



Containers and CI/CD

```
FROM node:12


WORKDIR /opt/app
COPY package.json package-lock.json ./
RUN npm ci
COPY ./app /opt/app
EXPOSE 80


CMD [ "node", "service.js" ]
```

Containers and CI/CD



Best practices for CI/CD

1. Automated releases

2. Safe deployments

3. Repeatable infrastructure changes

Best practices for CI/CD

1. Automated releases

2. Safe deployments

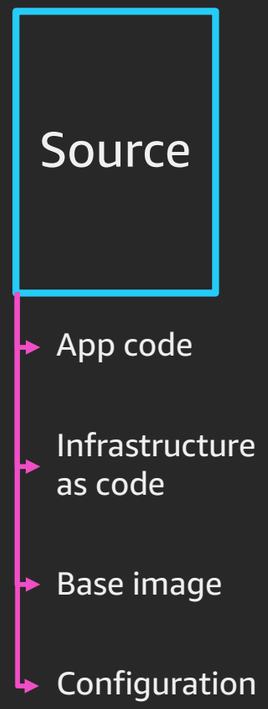
3. Repeatable infrastructure changes

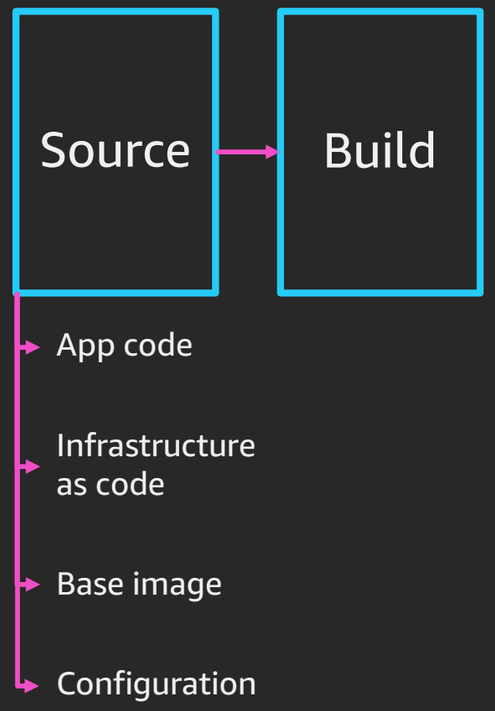
190 million deployments

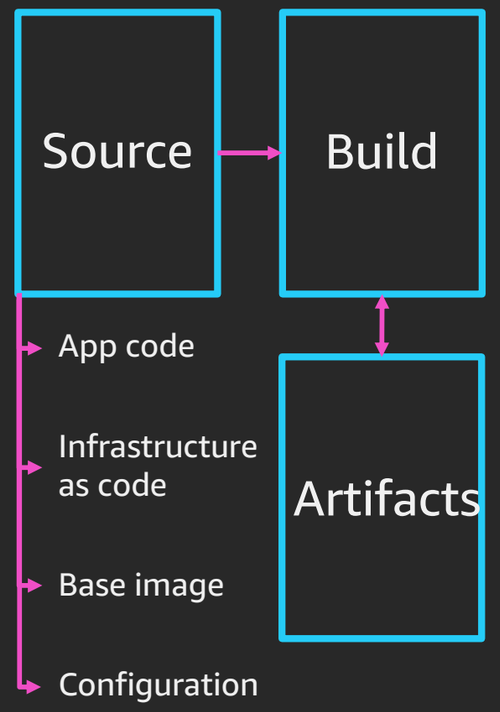
190 million deployments

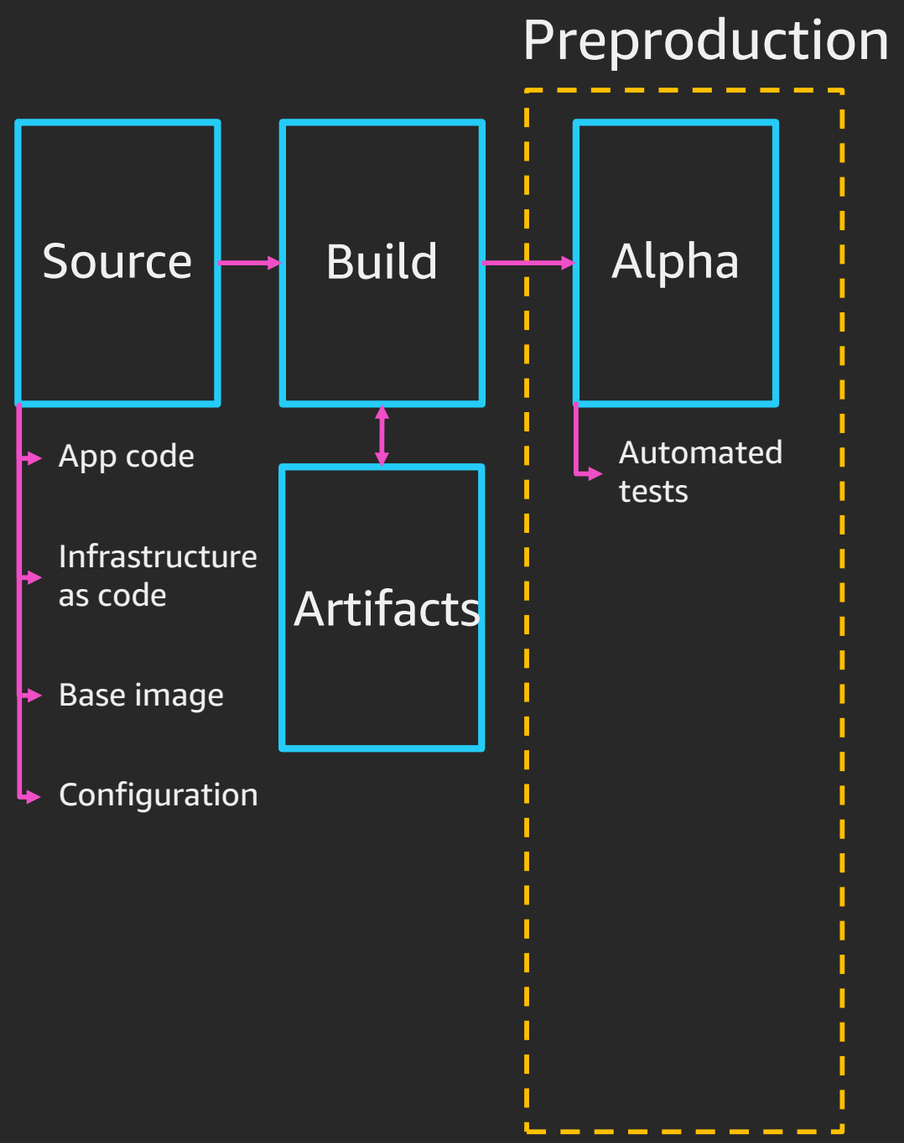
(6 deployments per second)

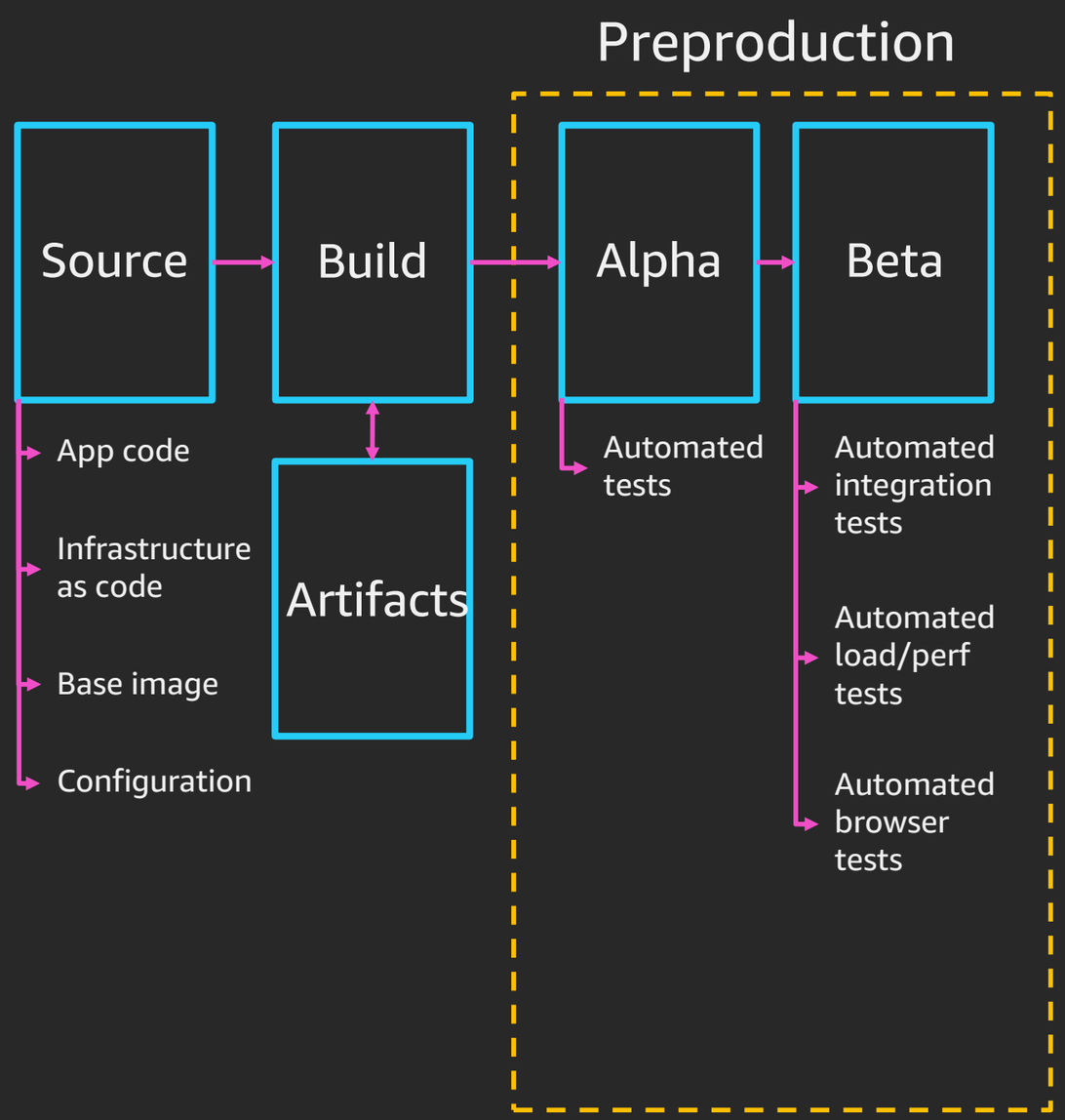
Source

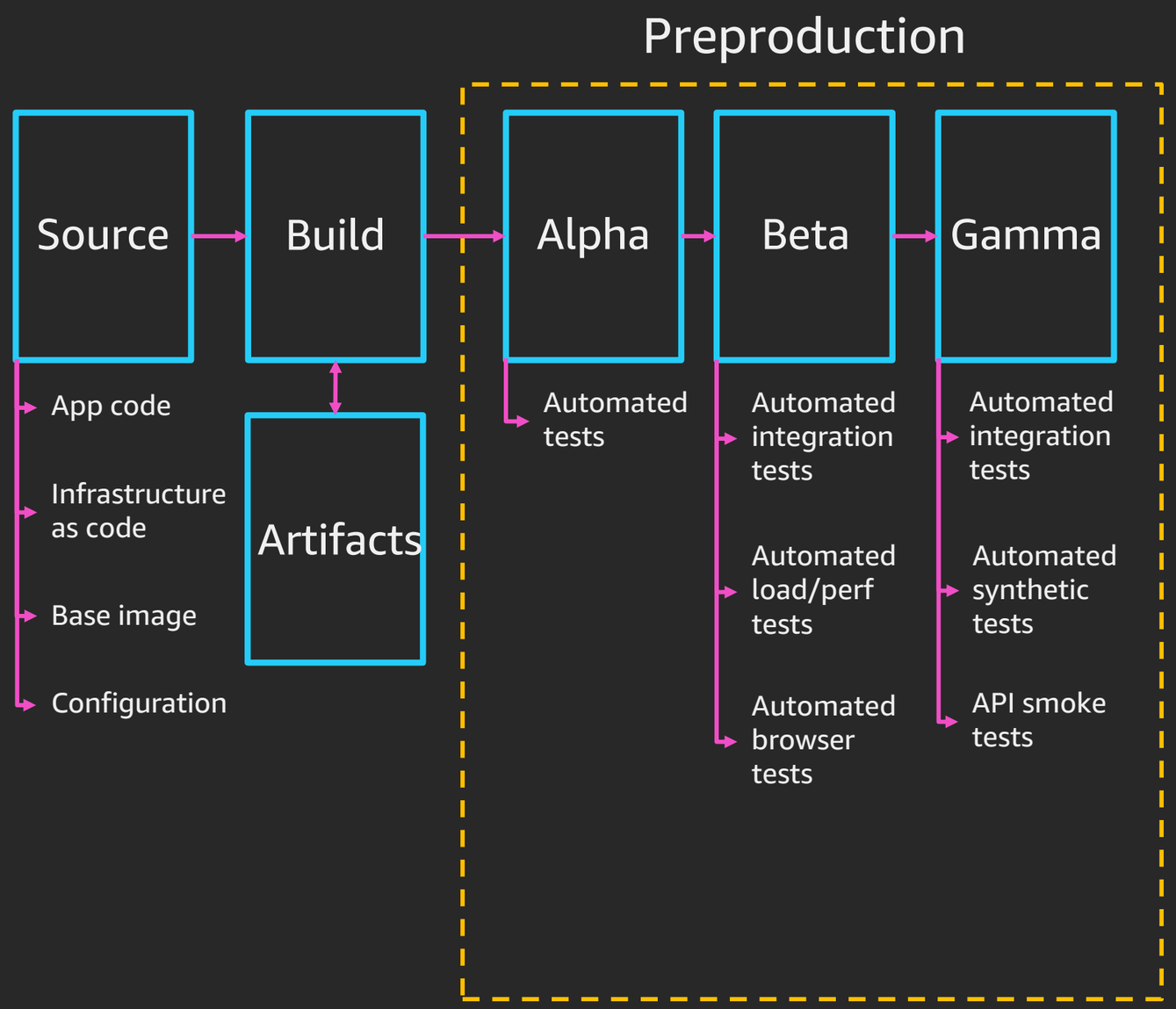

Source
App code
Infrastructure as code
Base image
Configuration

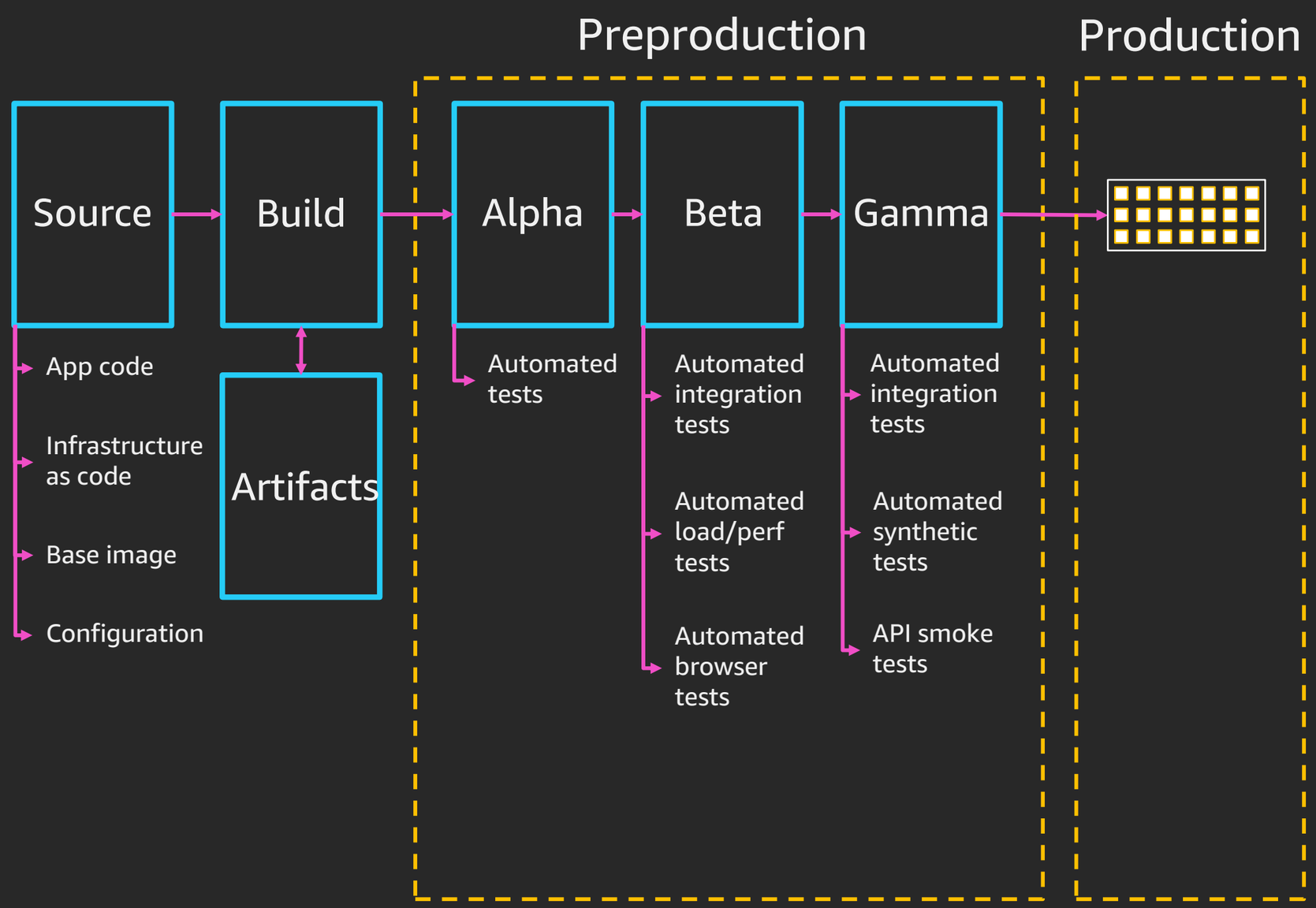

Source
Build
App code
Infrastructure as code
Base image
Configuration

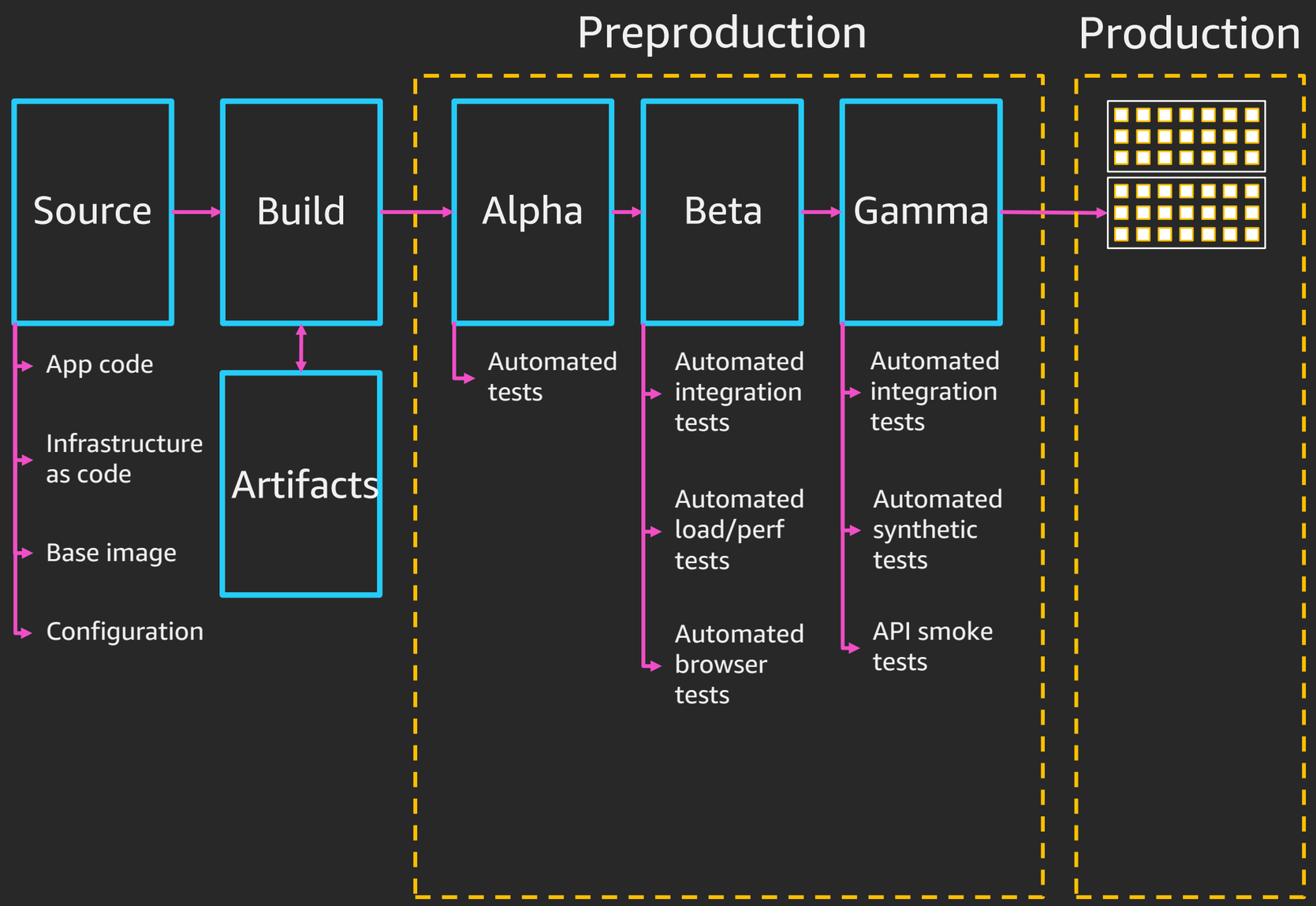

Source
Build
App code
Infrastructure as code
Base image
Configuration
Artifacts

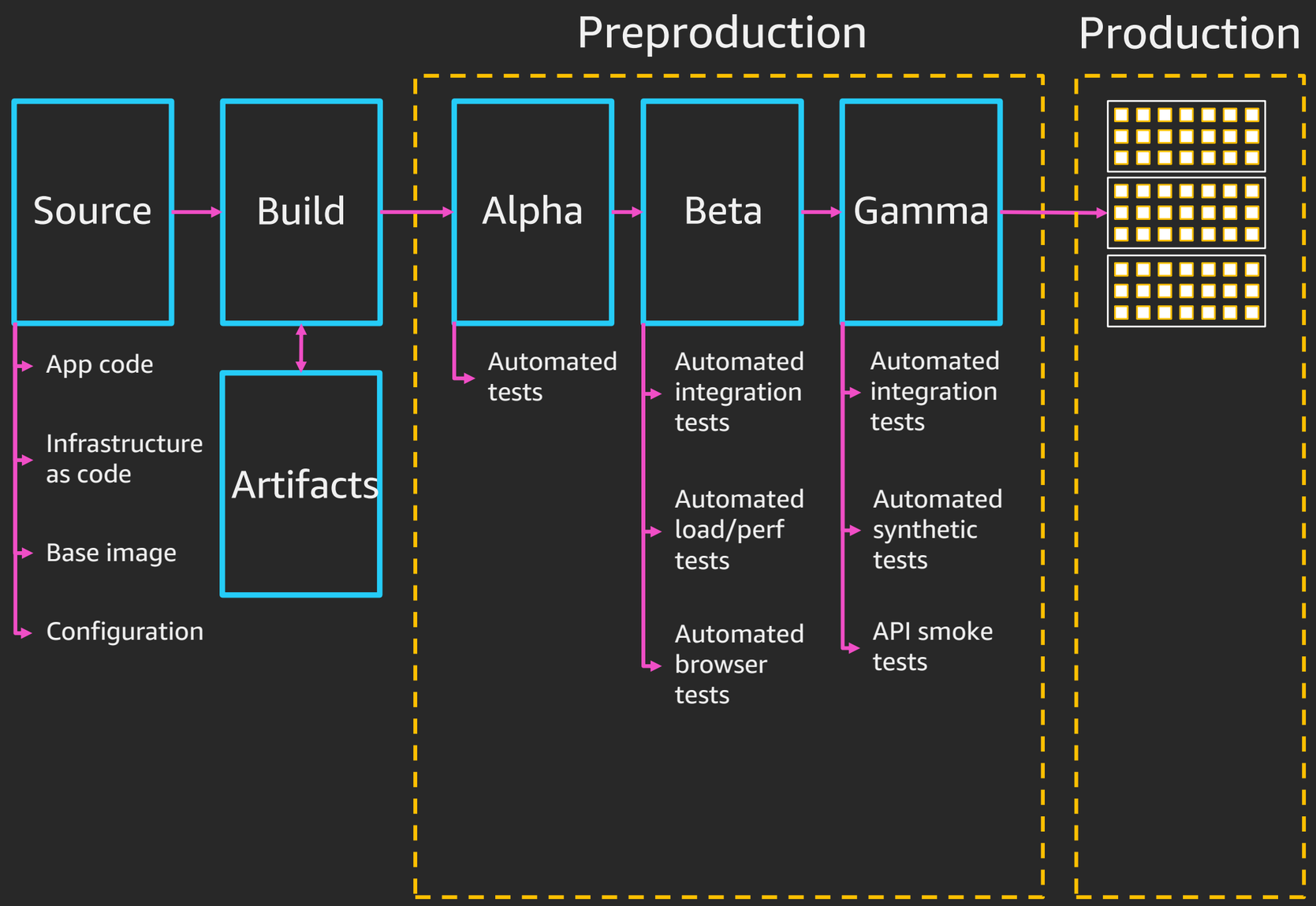

Preproduction
Source
Build
Alpha
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests

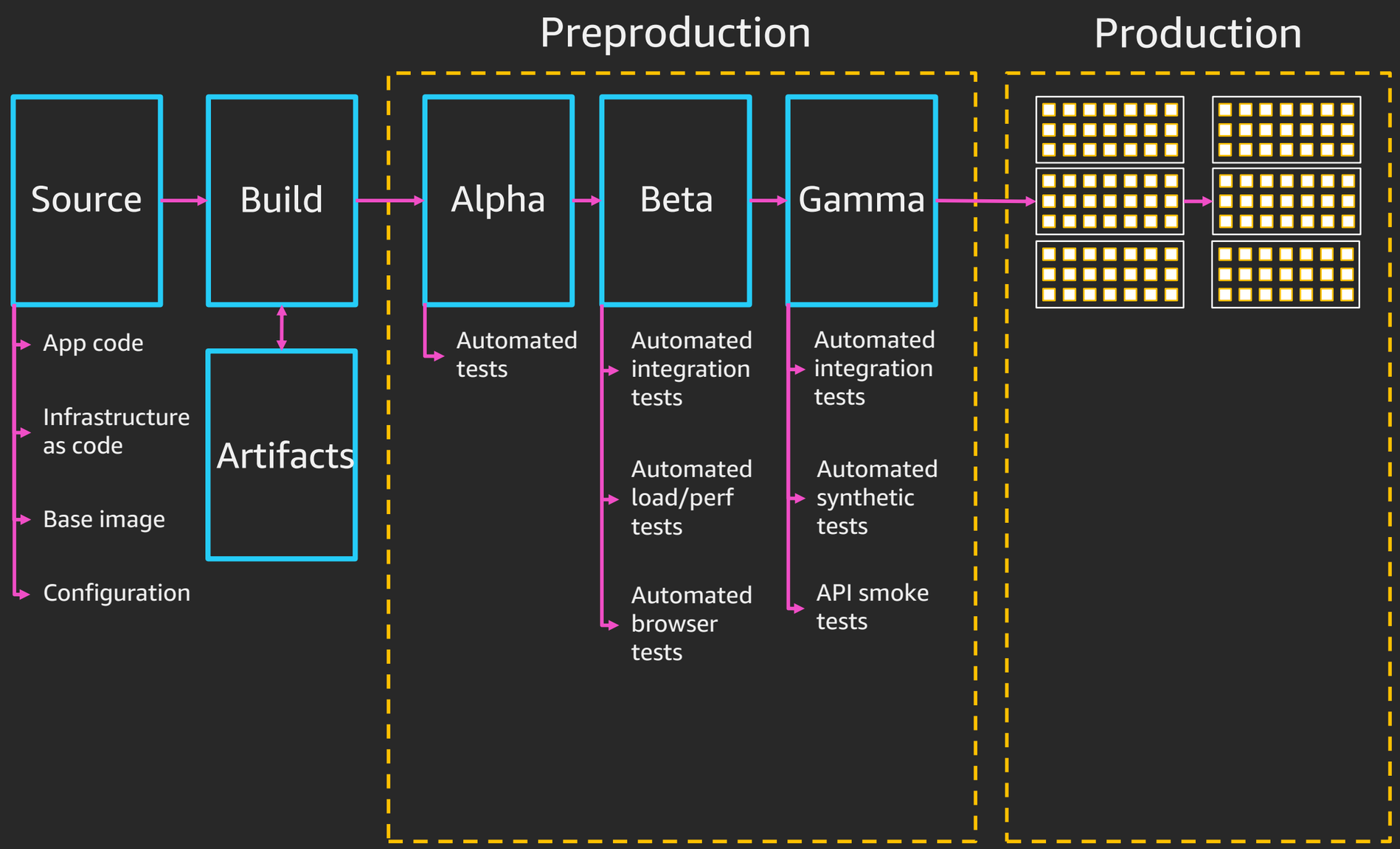

Preproduction
Source
Build
Alpha
Beta
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests

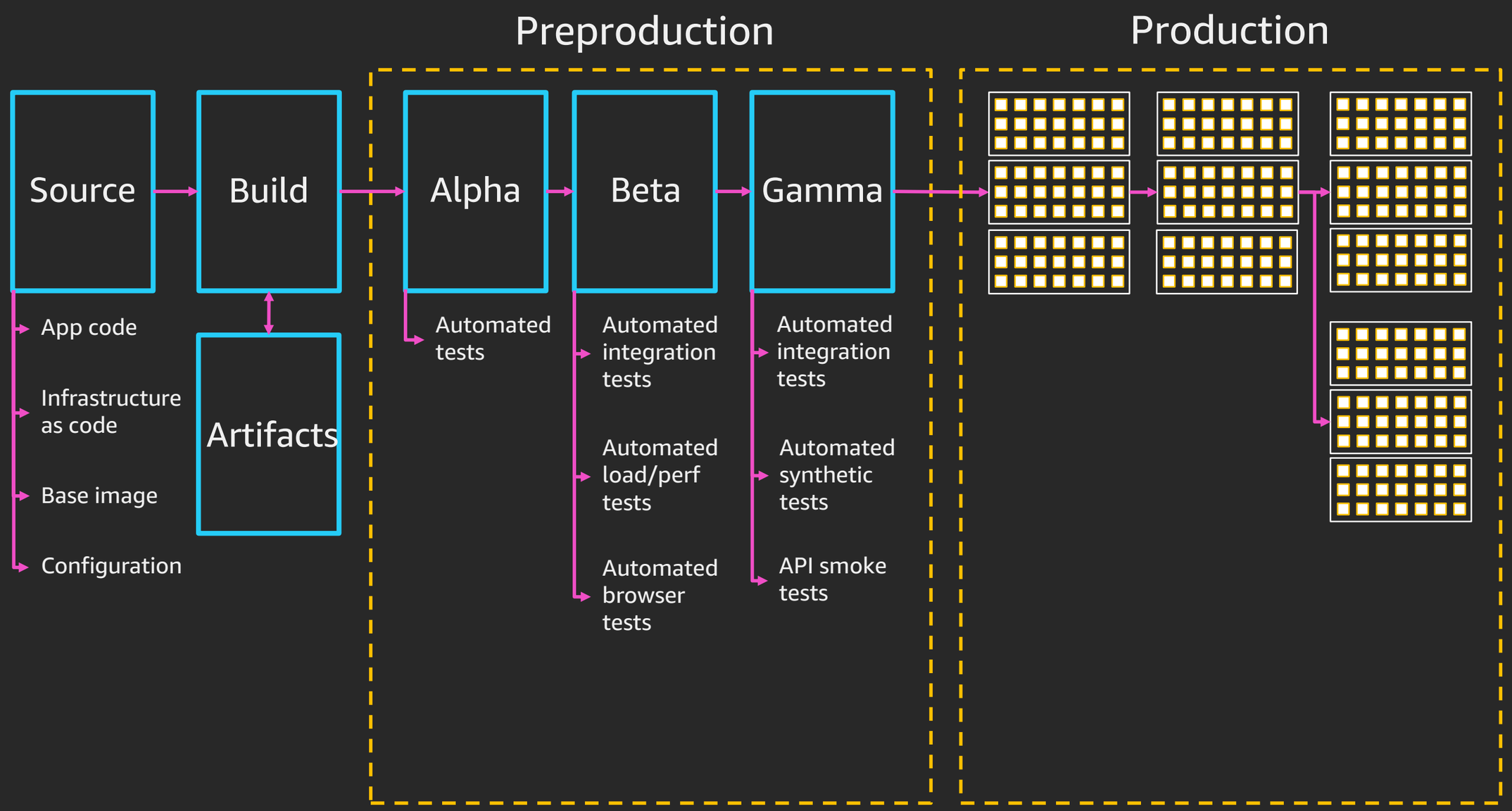

Preproduction
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests

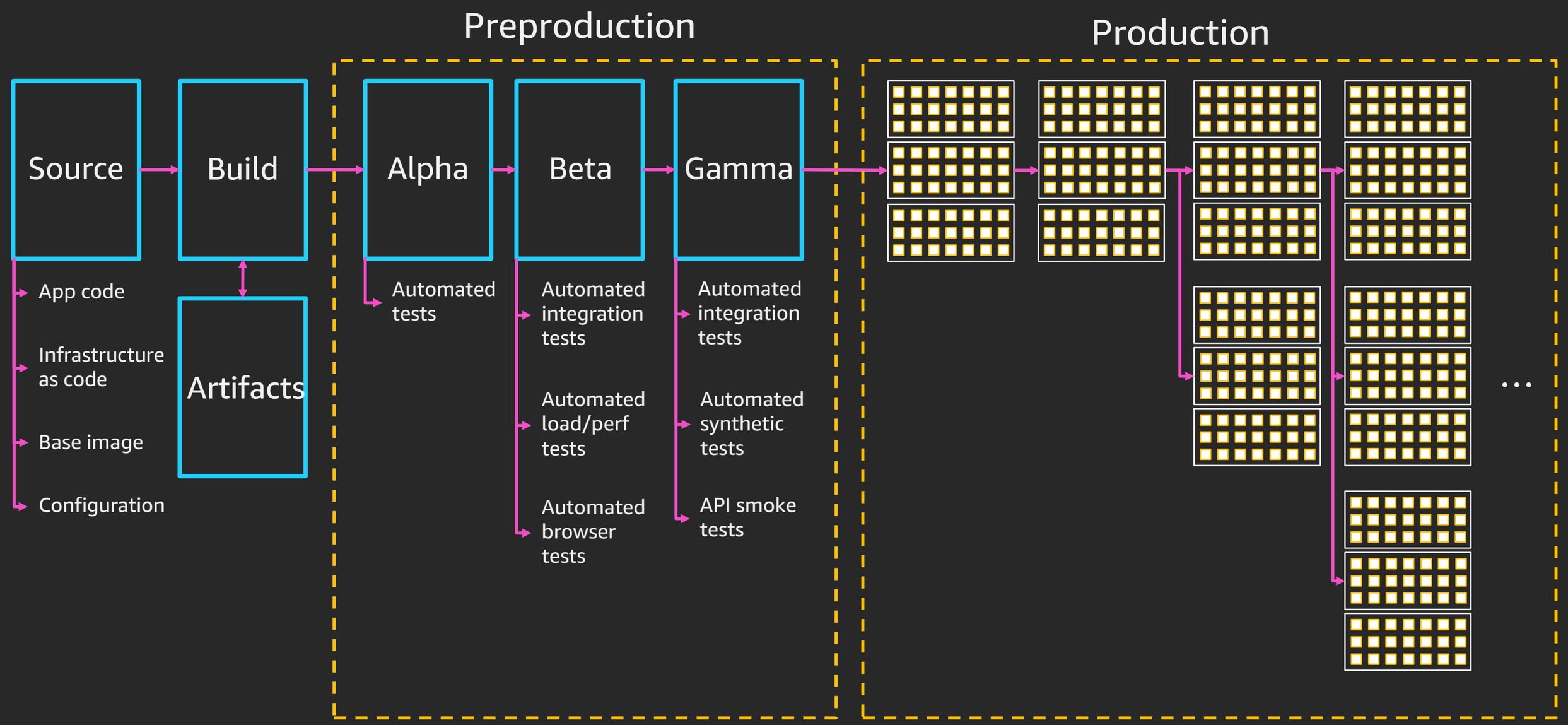

Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests

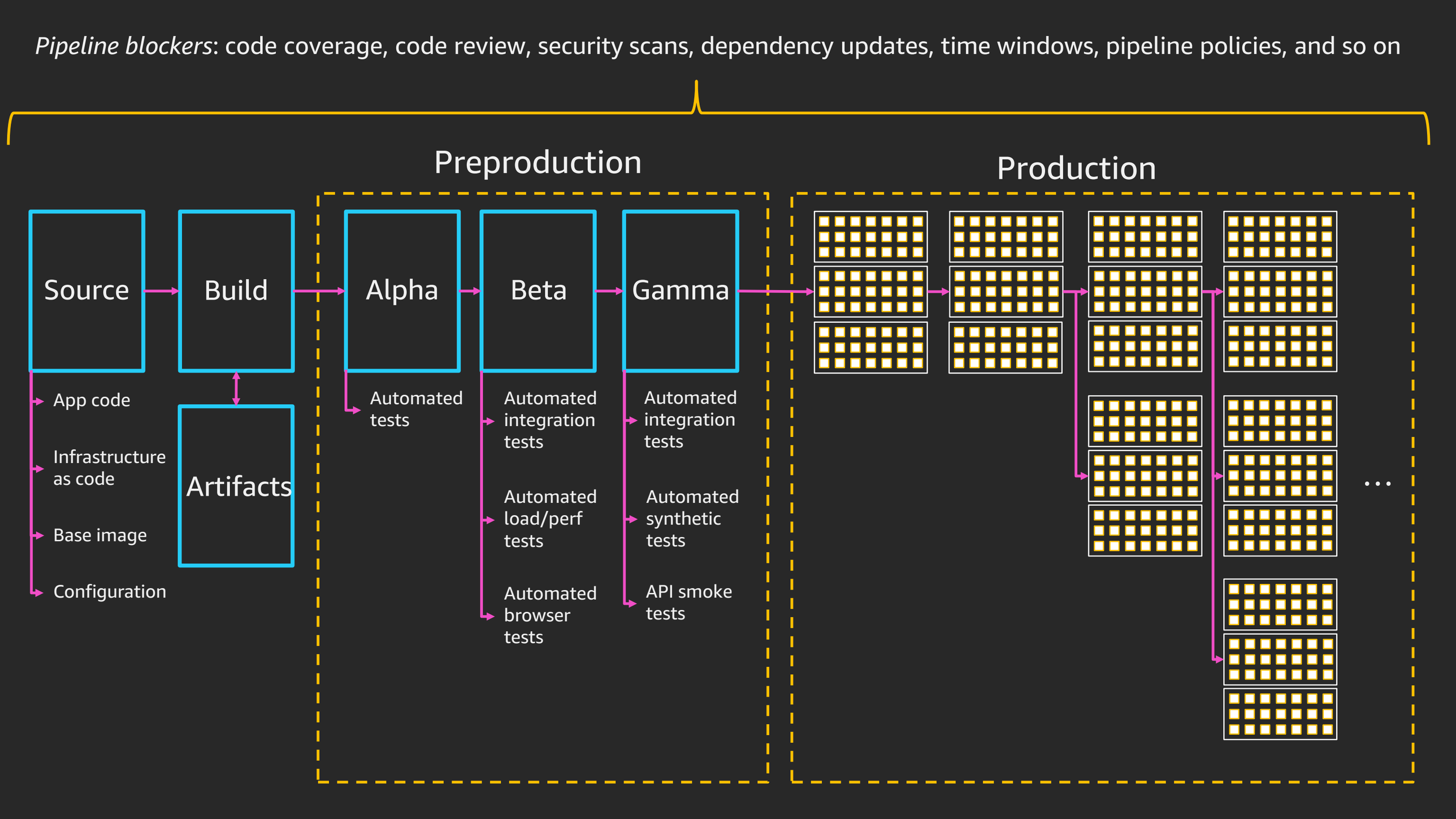

Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests


Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests


Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests


Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests


Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests
...


Pipeline blockers: code coverage, code review, security scans, dependency updates, time windows, pipeline policies, and so on
Preproduction
Production
Source
Build
Alpha
Beta
Gamma
App code
Infrastructure as code
Base image
Configuration
Artifacts
Automated tests
Automated integration tests
Automated load/perf tests
Automated browser tests
Automated integration tests
Automated synthetic tests
API smoke tests
...

Release process stages

Release process stages

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

Release process stages

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

- Compile code
- Unit tests
- Style checkers
- Create container images

Release process stages

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

- Compile code
- Unit tests
- Style checkers
- Create container images

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

Release process stages

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

- Compile code
- Unit tests
- Style checkers
- Create container images

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

- Deploy to production environments
- Monitor code in production in order to quickly detect errors

Release process stages

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

- Compile code
- Unit tests
- Style checkers
- Create container images

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

- Deploy to production environments
- Monitor code in production in order to quickly detect errors

CI/CD tools for Amazon ECS

AWS CodePipeline

GitHub Actions

Jenkins

Spinnaker

AWS CodePipeline

- Model and visualize your release process
- Builds, tests, and deploys your code every time code changes
- Integrates with third-party tools and AWS services, including Amazon ECS

AWS CodePipeline: Release process stages

AWS CodePipeline: Release process stages

AWS CodePipeline: Supported sources

Automatically kick off release and pull latest source code

Pick branch

AWS CodeCommit

GitHub

Pick object or folder

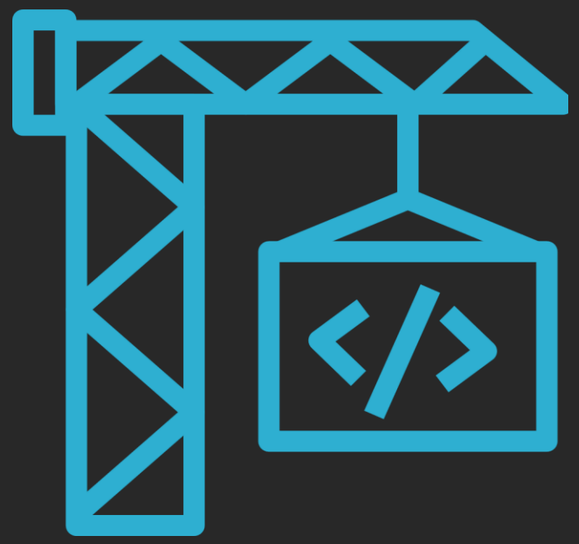
Amazon Simple Storage Service (Amazon S3)

Pick Docker tag

Amazon Elastic Container Registry (Amazon ECR)

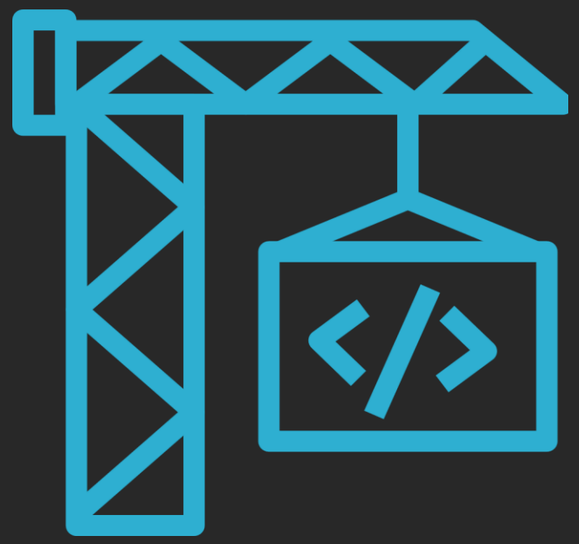
AWS CodePipeline: Release process stages

AWS CodeBuild



- Fully managed software build service
- Scales continuously and processes multiple builds concurrently
- No build servers to manage
- Pay by the minute, for only the compute resources you use

AWS CodeBuild for containers



- Build, push, and validate Docker images
- Docker and AWS Command Line Interface (AWS CLI) are installed in every official CodeBuild image
- Role credentials populated into build environment for pushing Docker images to Amazon ECR
- Provide custom build environment Docker image

AWS CodeBuild: Container buildspec

```
version: 0.2

phases:
  build:
    commands:
      - $(aws ecr get-login --no-include-email)
      - docker build -t $IMAGE_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_NAME:$IMAGE_TAG $ECR_REPO:$IMAGE_TAG
      - docker push $ECR_REPO:$IMAGE_TAG
```

Container image tagging

Container image tagging

Build pushes new “latest” image

AWS Fargate service

Image: sha256@11111...

Image: sha256@22222... (“latest”)

Amazon ECR repository

Container image tagging

Service scales up, launching new tasks

AWS Fargate service

Image: sha256@11111...

Image: sha256@22222... (“latest”)

Amazon ECR repository

AWS CodeBuild: Container image tagging

```
version: 0.2

phases:
  build:
    commands:
      - export IMAGE_TAG=build-`echo build-$CODEBUILD_BUILD_ID
          | awk -F":" ‘{print $2}‘`
      - $(aws ecr get-login --no-include-email)
      - docker build -t $IMAGE_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_NAME:$IMAGE_TAG $ECR_REPO:$IMAGE_TAG
      - docker push $ECR_REPO:$IMAGE_TAG
```

Amazon ECR immutable image tags


```
$ aws ecr put-image-tag-mutability \
      --repository-name my-ecr-repo \
      --image-tag-mutability IMMUTABLE
```

```
$ docker push $ECR_REPO:latest
```

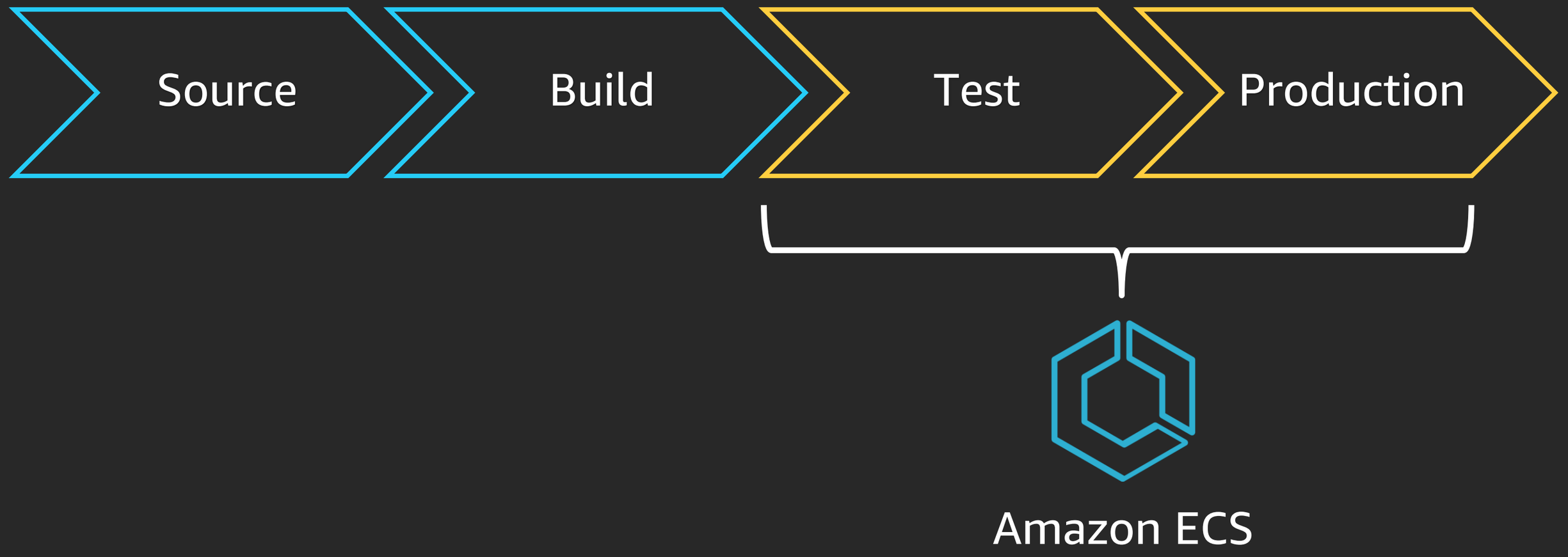
Tag invalid: The image tag ‘latest’ already exists in the ‘my-ecr-repo’ repository and cannot be overwritten because the repository is immutable.

Amazon ECR image scanning

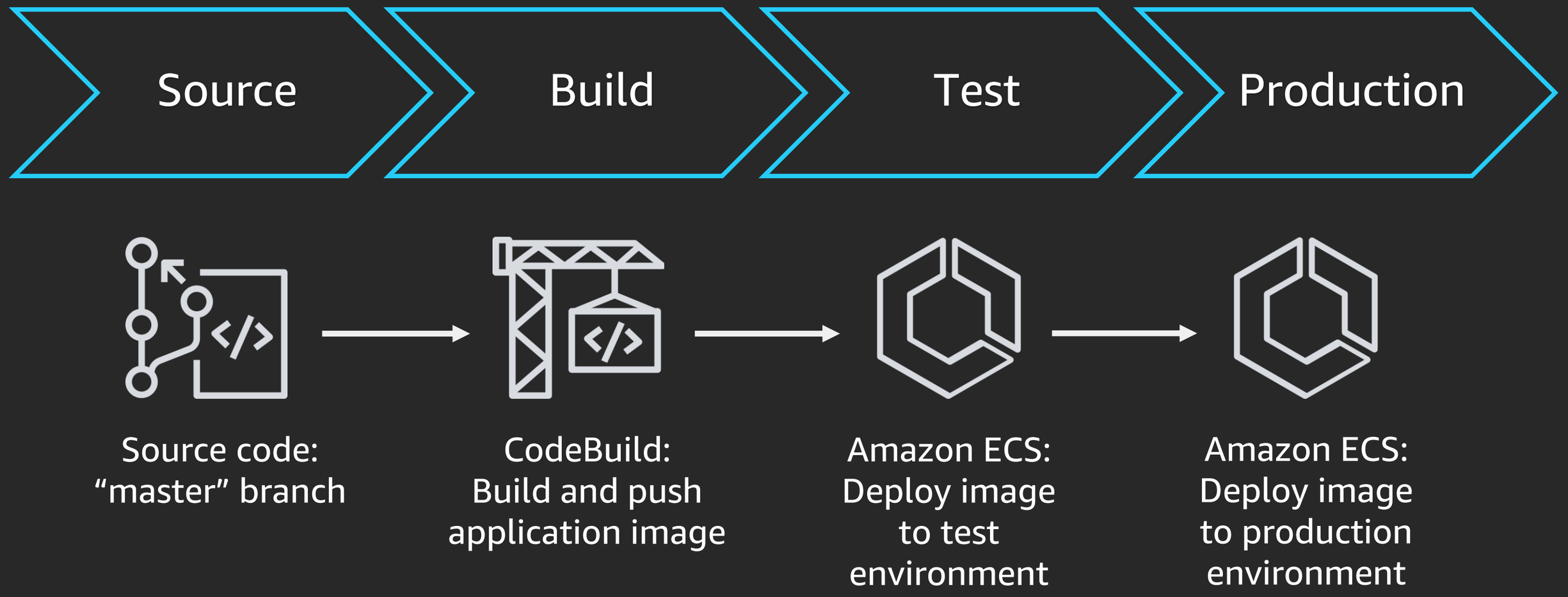

```
$ aws ecr put-image-scanning-configuration \
      --repository-name my-ecr-repo \
      --image-scanning-configuration scanOnPush=true

$ docker push $ECR_REPO:$BUILD_ID_TAG

$ aws ecr describe-image-scan-findings \
      --repository-name my-ecr-repo \
      --image-id imageTag=$BUILD_ID_TAG
```

AWS CodePipeline: Release process stages

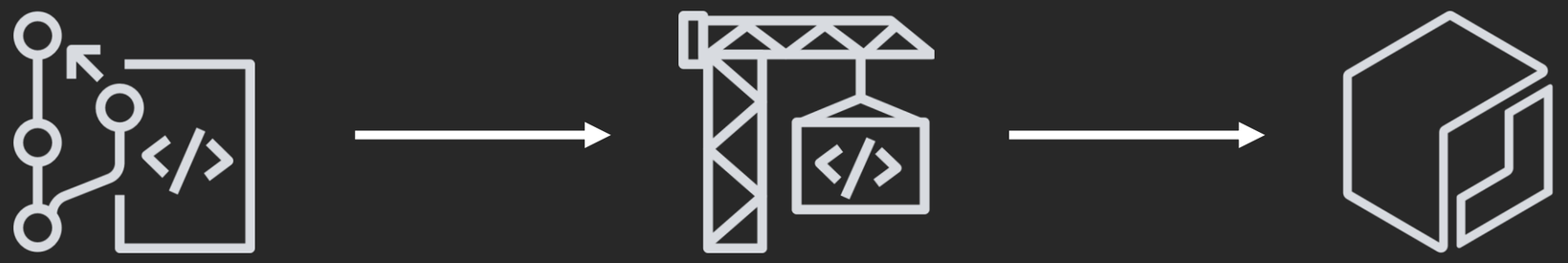
AWS CodePipeline: Release process stages



AWS CodePipeline: Sample pipeline



AWS CodePipeline: Image pipeline

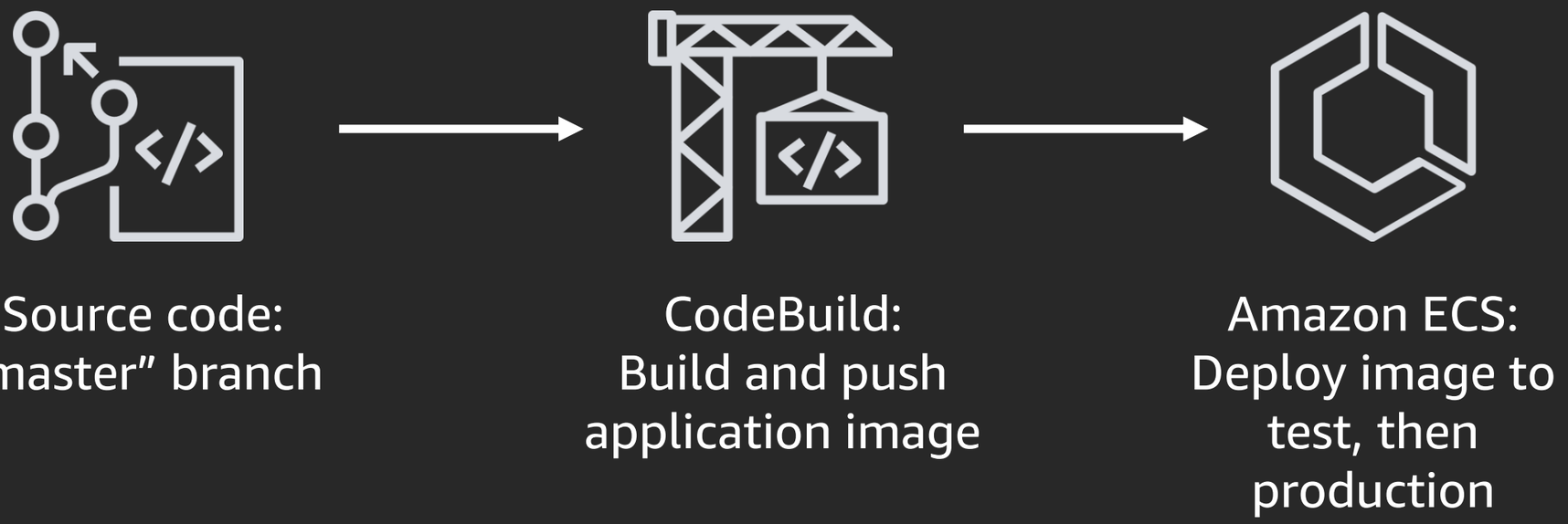


Source code:
“master” branch

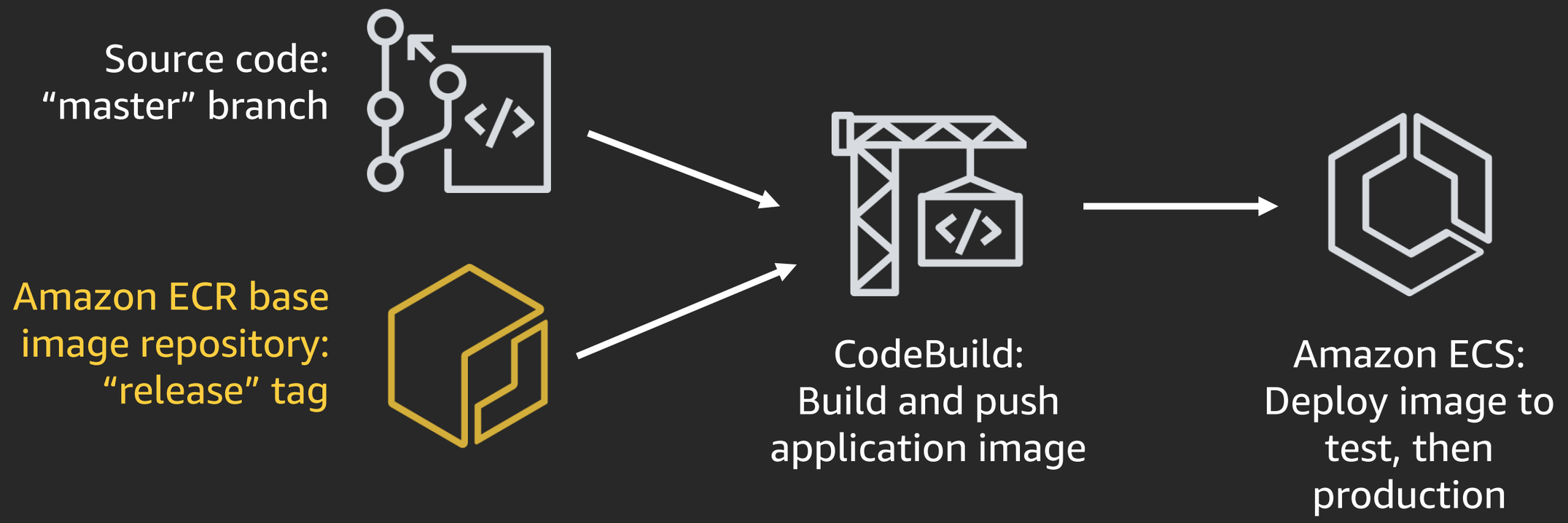
CodeBuild:
Build and push base image or sidecar image

Amazon ECR base image or sidecar image repository:
“release” tag

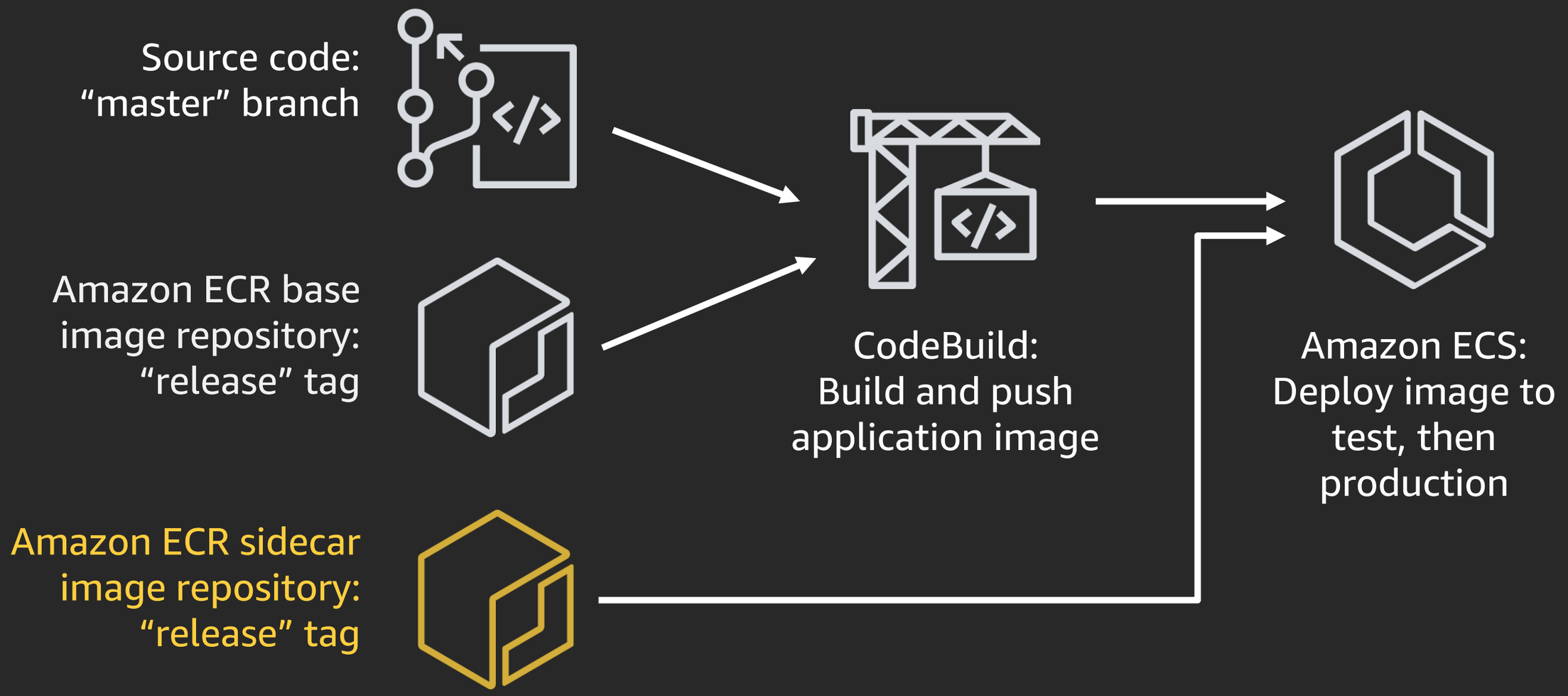
AWS CodePipeline: Sample pipeline



AWS CodePipeline: Sample pipeline



AWS CodePipeline: Sample pipeline



Release process stages

GitHub Actions workflow


```yaml
steps:
  - name: Checkout
    uses: actions/checkout@v1
  - name: Configure AWS credentials
    uses: aws-actions/configure-aws-credentials@v1
    with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: us-east-2
  - name: Deploy Amazon ECS task definition
    uses: aws-actions/amazon-ecs-deploy-task-definition@v1
    with:
        task-definition: task-definition.json
        service: my-ecs-service
        cluster: my-ecs-cluster
```

Jenkinsfile

```
stage('Deploy to ECS Test Service') {
  steps {
    script {
      sh '''#!/bin/bash -ex
         sed -i s/BUILD/${BUILD_NUMBER}/g taskdef.json
         REV=$(aws ecs register-task-definition --cli-input-json file://taskdef.json | \
              jq '.taskDefinition.taskDefinitionArn')
         aws ecs update-service --cluster ${CLUSTER} --service ${APP} \
              --task-definition ${REV}
         aws ecs wait services-stable --cluster ${ECS_CLUSTER} --services ${ECS_SERVICE}
       '''
     }
   }
}
```

Spinnaker pipeline


```
"stages": [
  {
        "name": "Deploy to Test ECS Service",
        "type": "deploy"
        "clusters": [
          {
                “cloudProvider": “ecs",
                "strategy": "redblack",
                "rollback": { "onFailure": true },
                "ecsClusterName": “test-cluster",
                "imageDescription": {
                  "account": "my-ecr-registry",
                  "fromTrigger": true,
                  "registry": "123456789012.dkr.ecr.eu-central-1.amazonaws.com",
                  "repository": "spinnaker-deployment-images"
```

Best practices for CI/CD

1. Automated releases

2. Safe deployments

3. Repeatable infrastructure changes

Best practices for CI/CD

1.

Automated releases

2.

Safe deployments

3.

Repeatable infrastructure changes

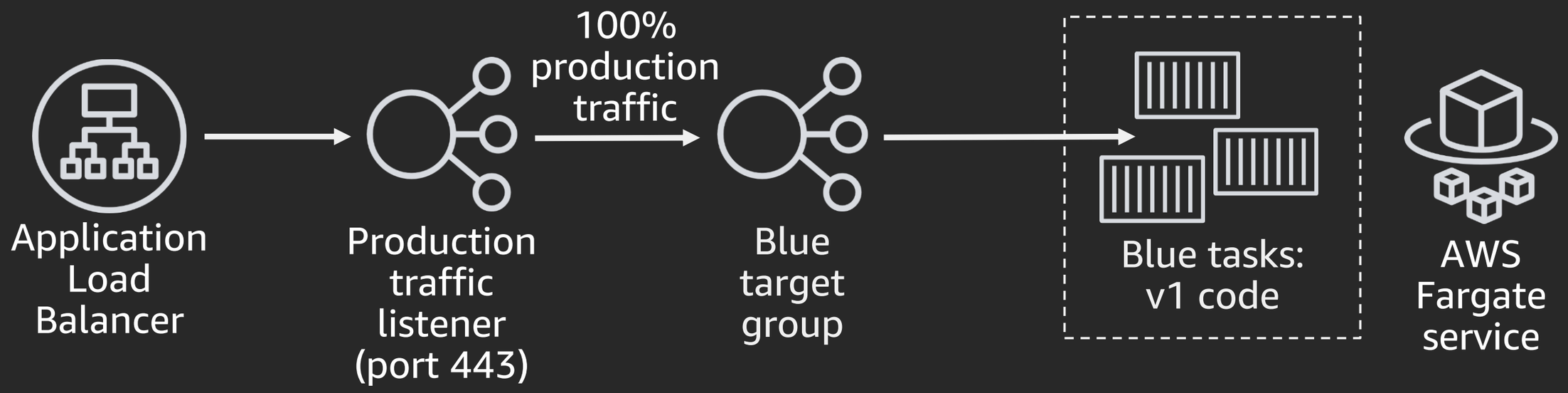
Amazon's deployment safety bar for CI/CD

1. Roll back *automatically* on alarms & validation tests
2. Roll back *quickly*
3. "*Bake*" after deployment
4. Deploy *small* at first, then more broadly

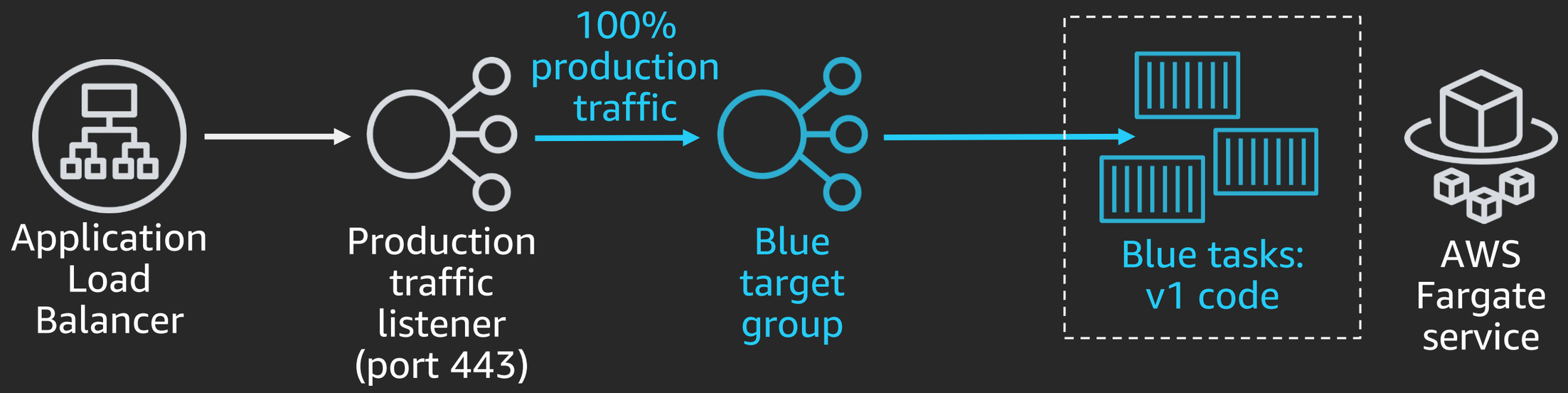
AWS CodeDeploy

- Automates code deployments to instances, ECS services, and Lambda functions
- Avoid downtime during application deployment
- Roll back automatically if failure is detected
- Limit customer impact with traffic control

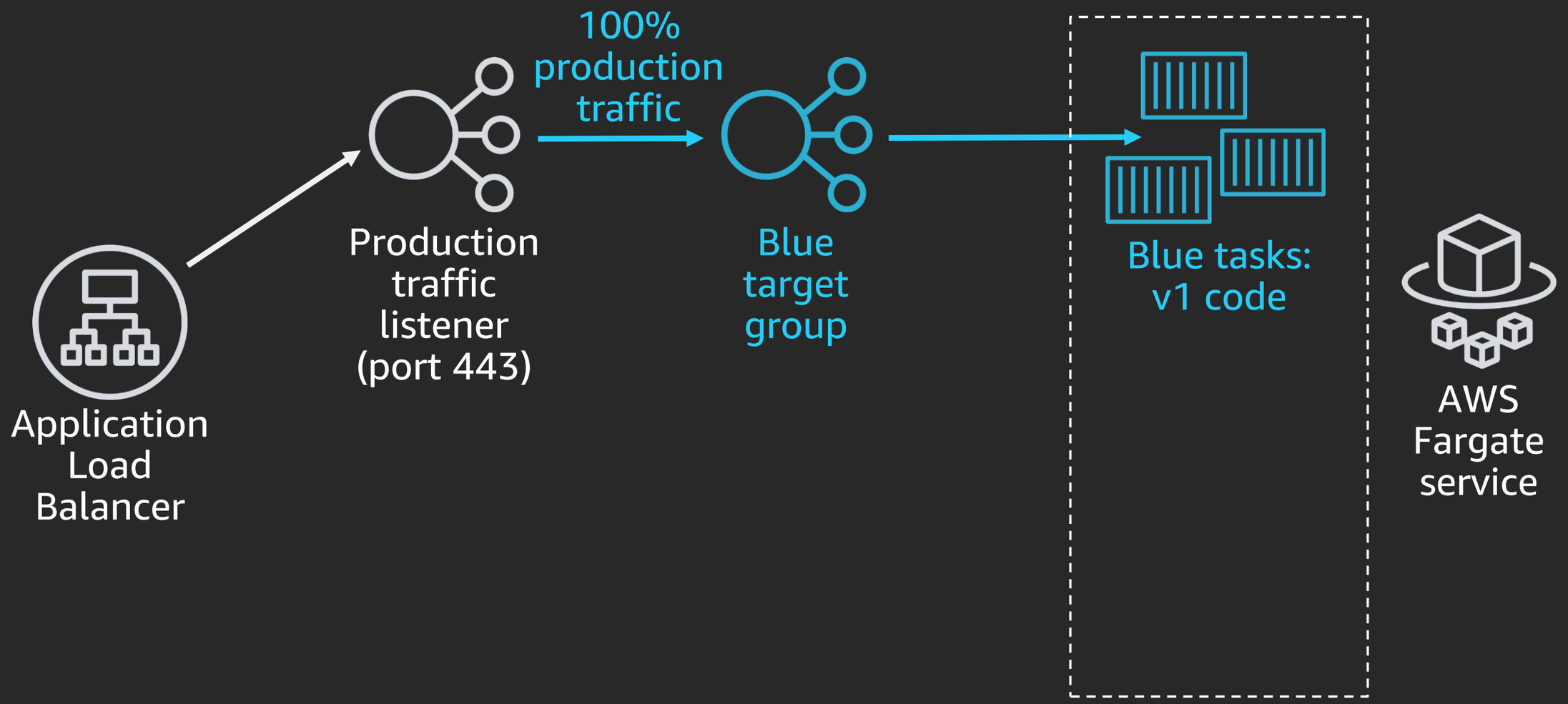
CodeDeploy ECS blue-green deployment



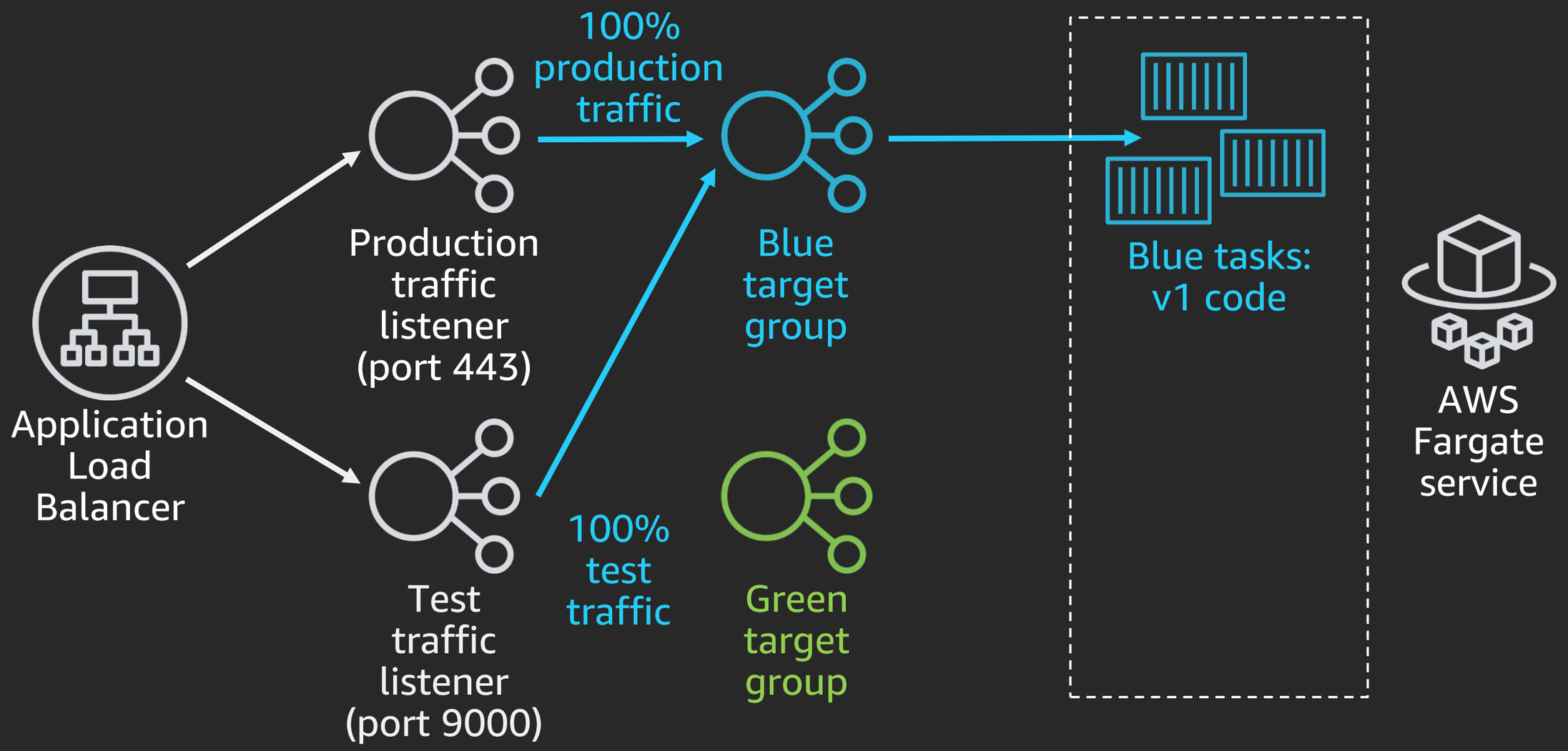
CodeDeploy ECS blue-green deployment



CodeDeploy ECS blue-green deployment

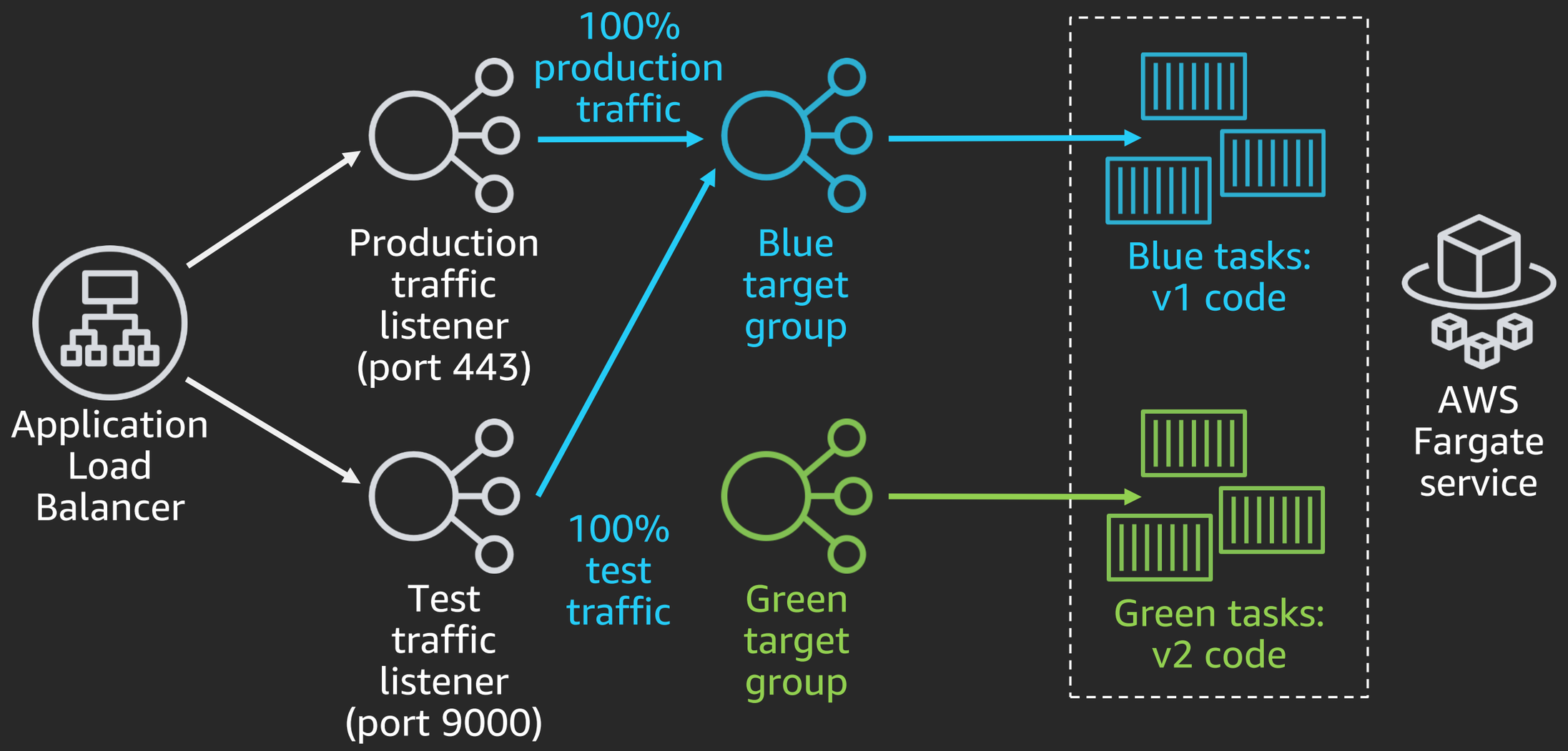


CodeDeploy ECS blue-green deployment



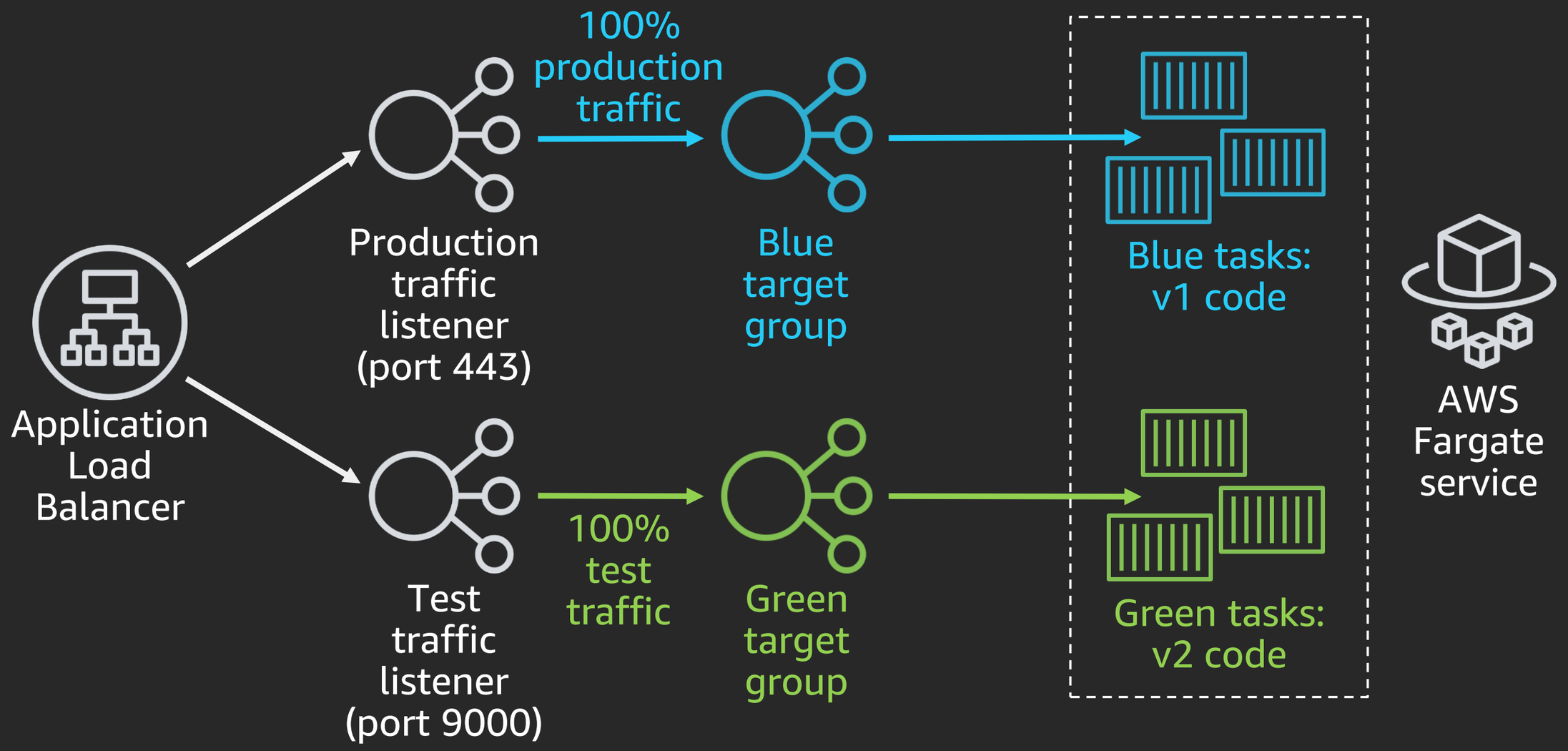
CodeDeploy ECS blue-green deployment

Provision green tasks



CodeDeploy ECS blue-green deployment

Shift test traffic to green; run validation tests against test endpoint



CodeDeploy ECS AppSpec file

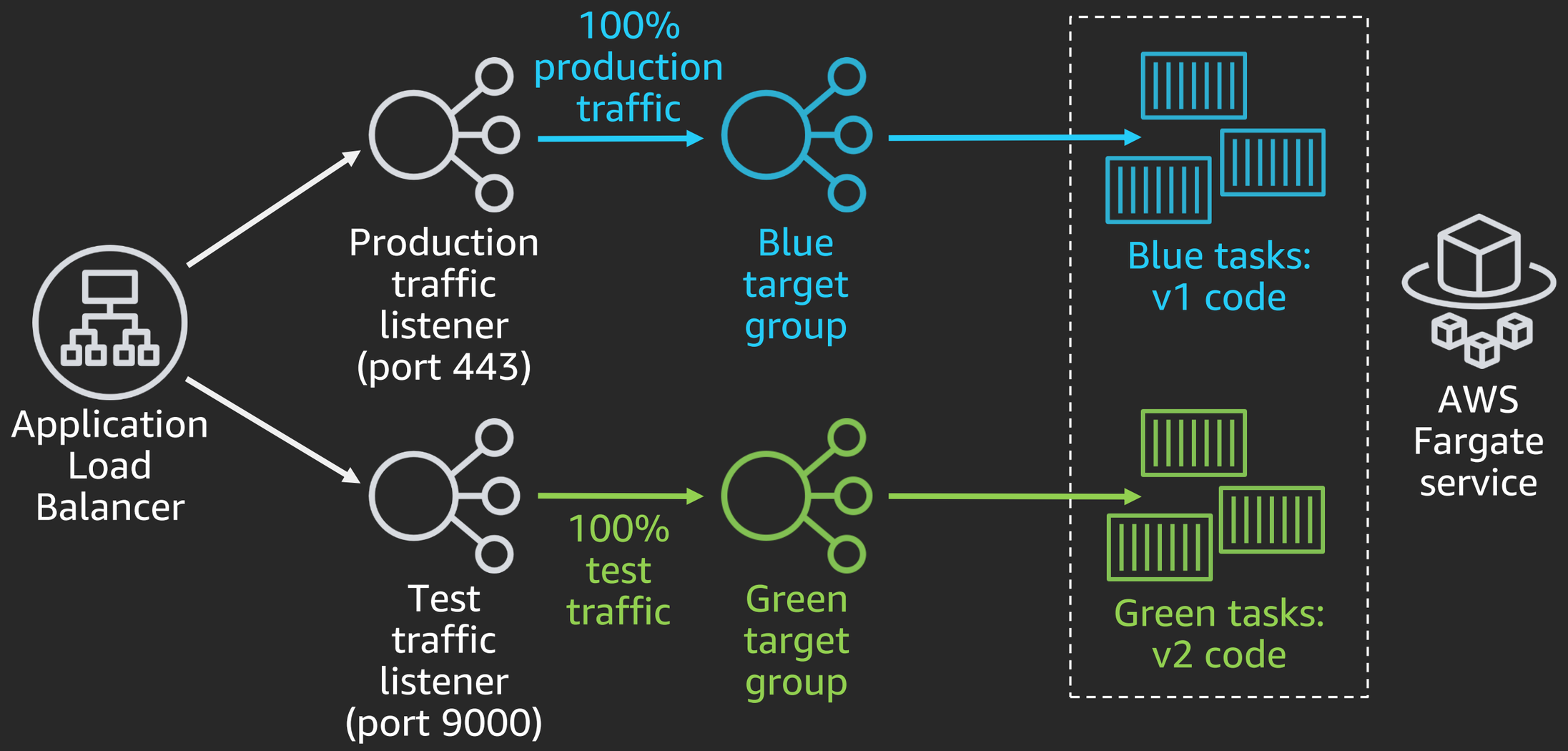
```
version: 1.0
Hooks:
    - BeforeInstall: "LambdaFunctionToExecuteAnythingBeforeNewRevisionInstalltion"
    - AfterInstall: "LambdaFunctionToExecuteAnythingAfterNewRevisionInstallation"
    - AfterAllowTestTraffic: "LambdaFunctionToValidateAfterTestTrafficShift"
    - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeTrafficShift"
    - AfterAllowTraffic: "LambdaFunctionToValidateAfterTrafficShift"
Resources:
    - TargetService:
        Type: AWS::ECS::Service
        Properties:
            - TaskDefinition: "my_task_definition:8"
              LoadBalancerInfos:
                    - ContainerName: "SampleApp"
                      ContainerPort: 80
```

CodeDeploy lifecycle hook

```
exports.handler = async function (event, context, callback) {
    var params = {
        deploymentId: event.DeploymentId,
        lifecycleEventHookExecutionId: event.LifecycleEventHookExecutionId,
        status: 'Succeeded'
    };

    const response = await axios(http://my-service.com:9000/api);
    if (response.status != 200) {
        params.status = 'Failed';
    }

    await codedeploy.putLifecycleEventHookExecutionStatus(params).promise();
}
```

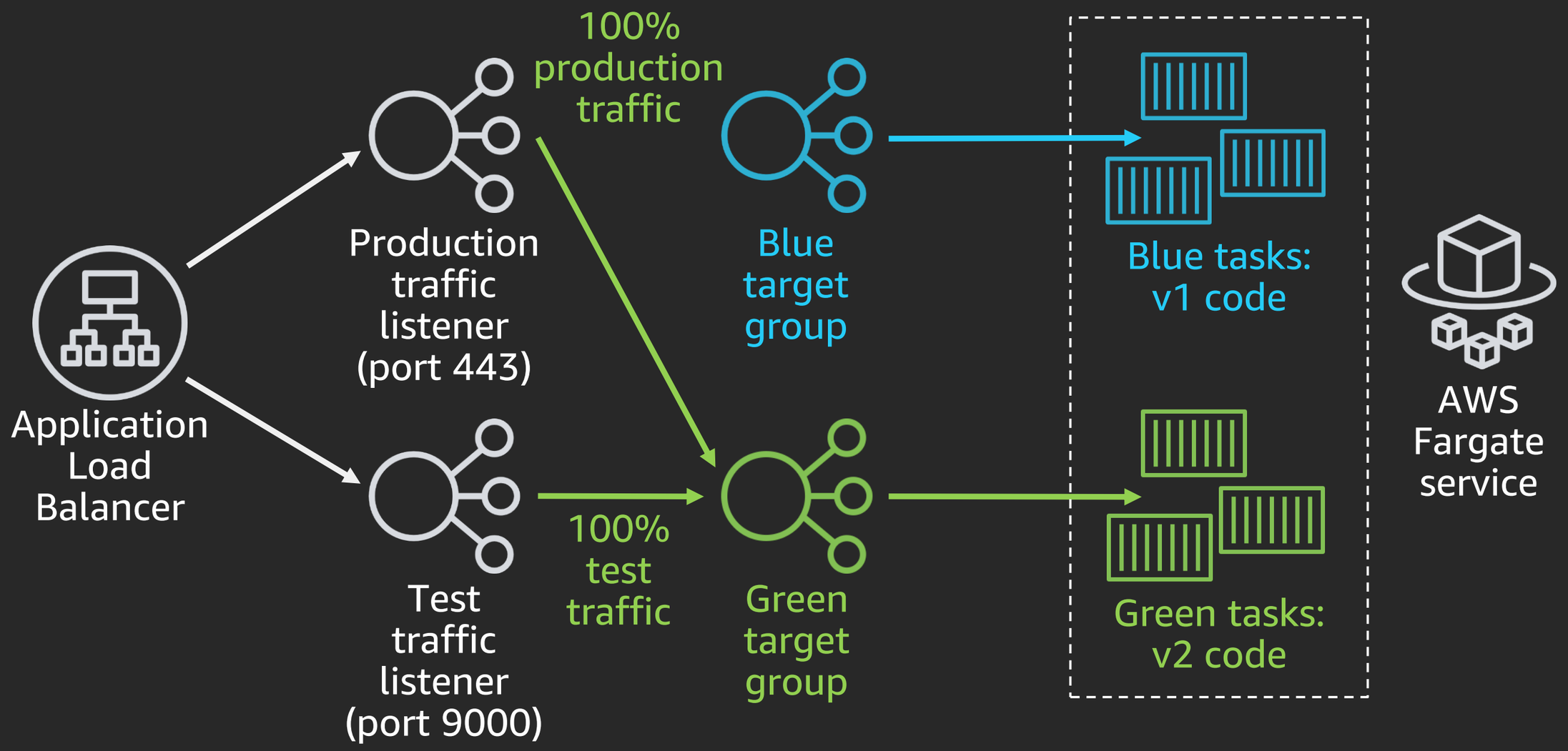
CodeDeploy ECS blue-green deployment

Shift test traffic to green, run validation tests against test endpoint



CodeDeploy ECS blue-green deployment

Shift production traffic to green; roll back in case of alarm



CodeDeploy deployment group configuration

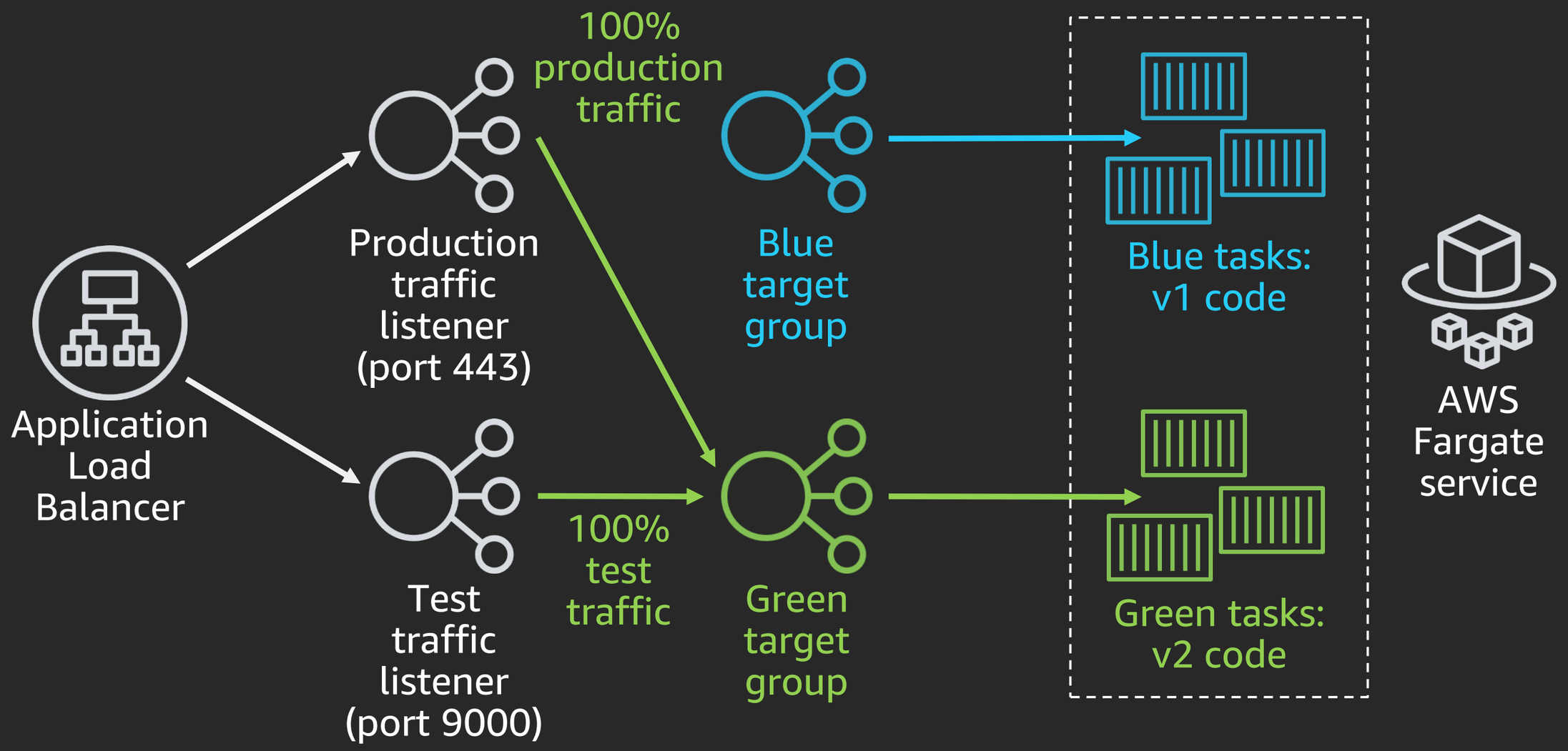
```
"alarmConfiguration": {
        "enabled": true,
        "ignorePollAlarmFailure": false,
        "alarms": [
                {
                        "name": "MyCloudWatchAlarm_Http5xx"
                },
                {
                        "name": "MyCloudWatchAlarm_UnhealthyHosts"
                },
                {
                        "name": "MyCloudWatchAlarm_ErrorLogging"
                }
        ]
}
```

CodeDeploy deployment group configuration

```
"blueGreenDeploymentConfiguration": {
        "terminateBlueInstancesOnDeploymentSuccess": {
                "action": "TERMINATE",
                "terminationWaitTimeInMinutes": 60
        }
},
```

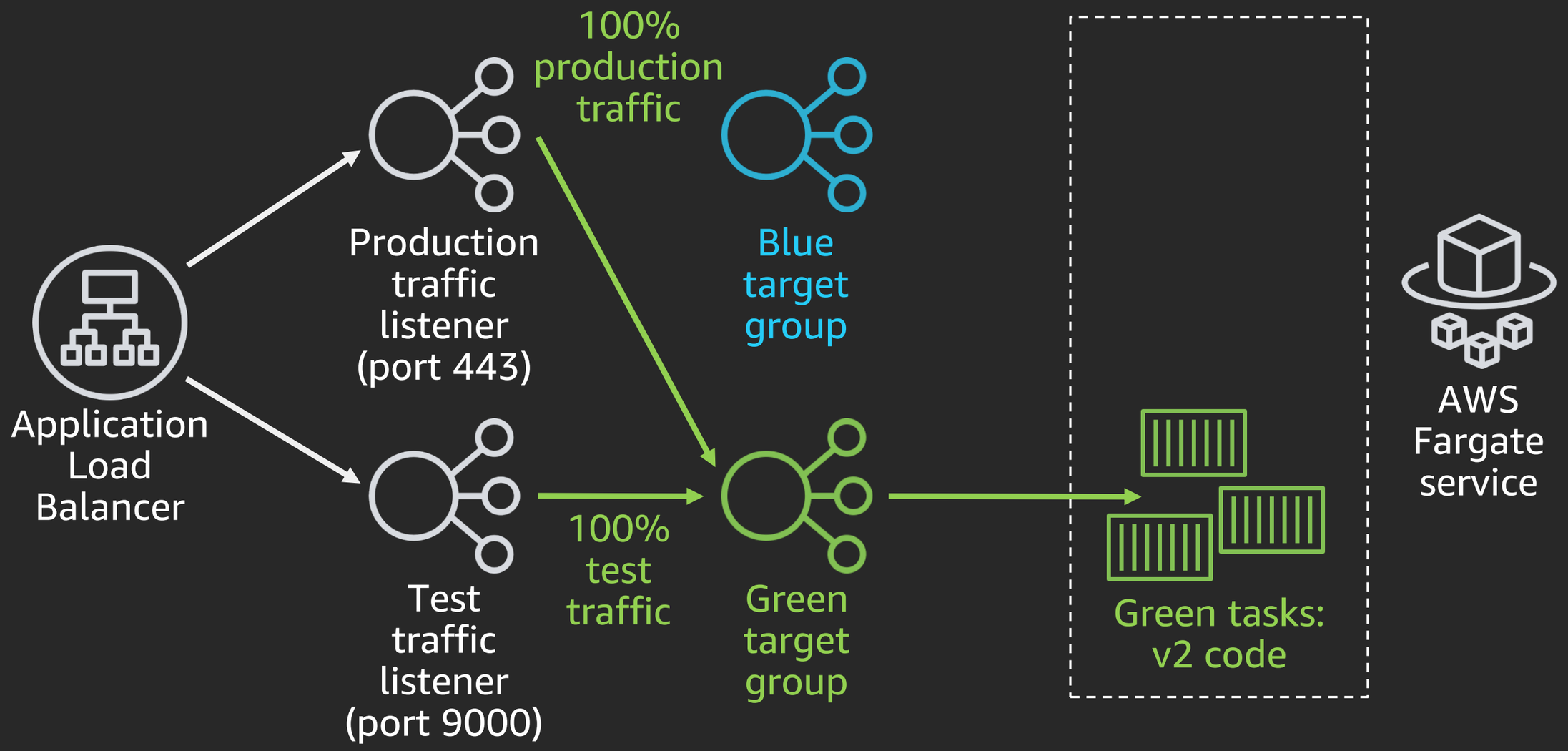
CodeDeploy ECS blue-green deployment

Shift production traffic to green; roll back in case of alarm



CodeDeploy ECS blue-green deployment

Drain blue tasks



CodeDeploy ECS blue-green deployment

- Use “CodeDeploy-ECS” deploy action in CodePipeline
- Use “aws ecs deploy” command in Jenkins and other CI/CD automation

```
aws ecs deploy \
     --service MyEcsService \
     --codedeploy-deployment-group MyDeploymentGroup \
     --task-definition task-definition.json \
     --codedeploy-appspec appspec.yml
```

Best practices for CI/CD

1.

Automated releases

2.

Safe deployments

3.

Repeatable infrastructure changes

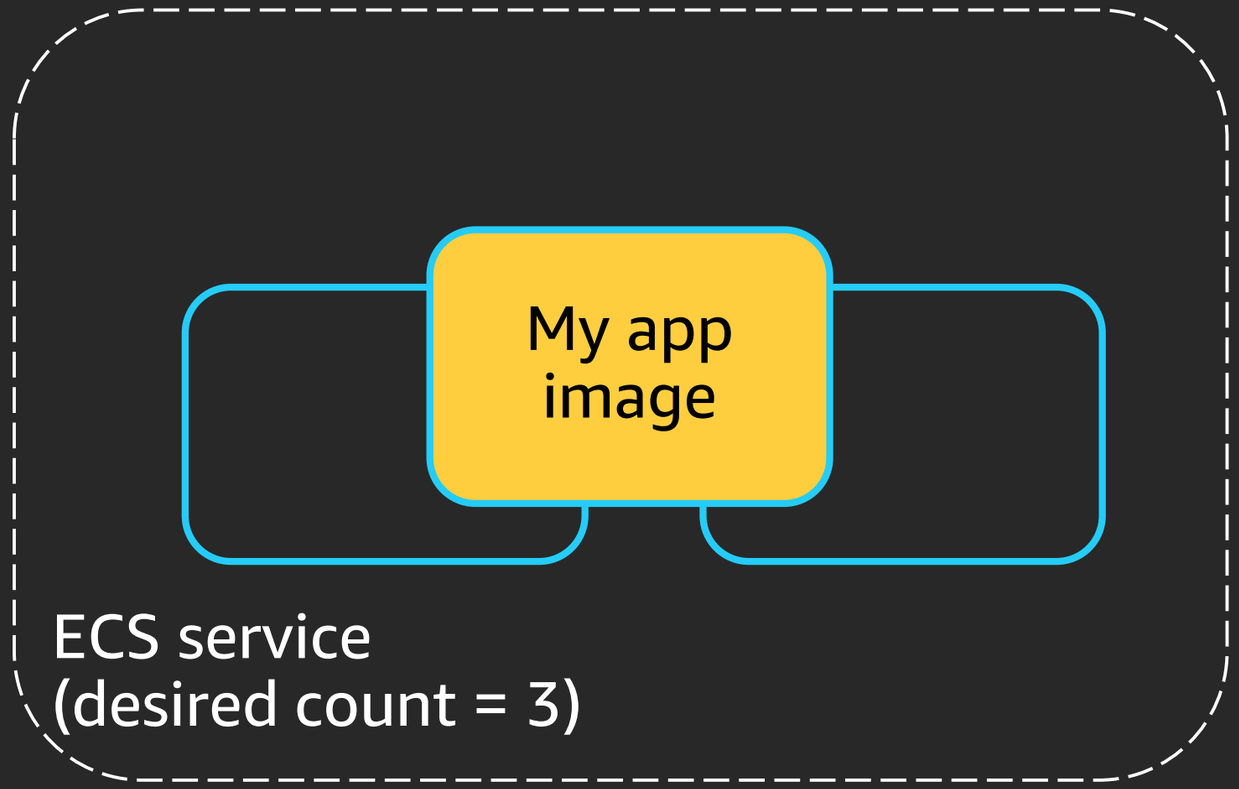
Best practices for CI/CD

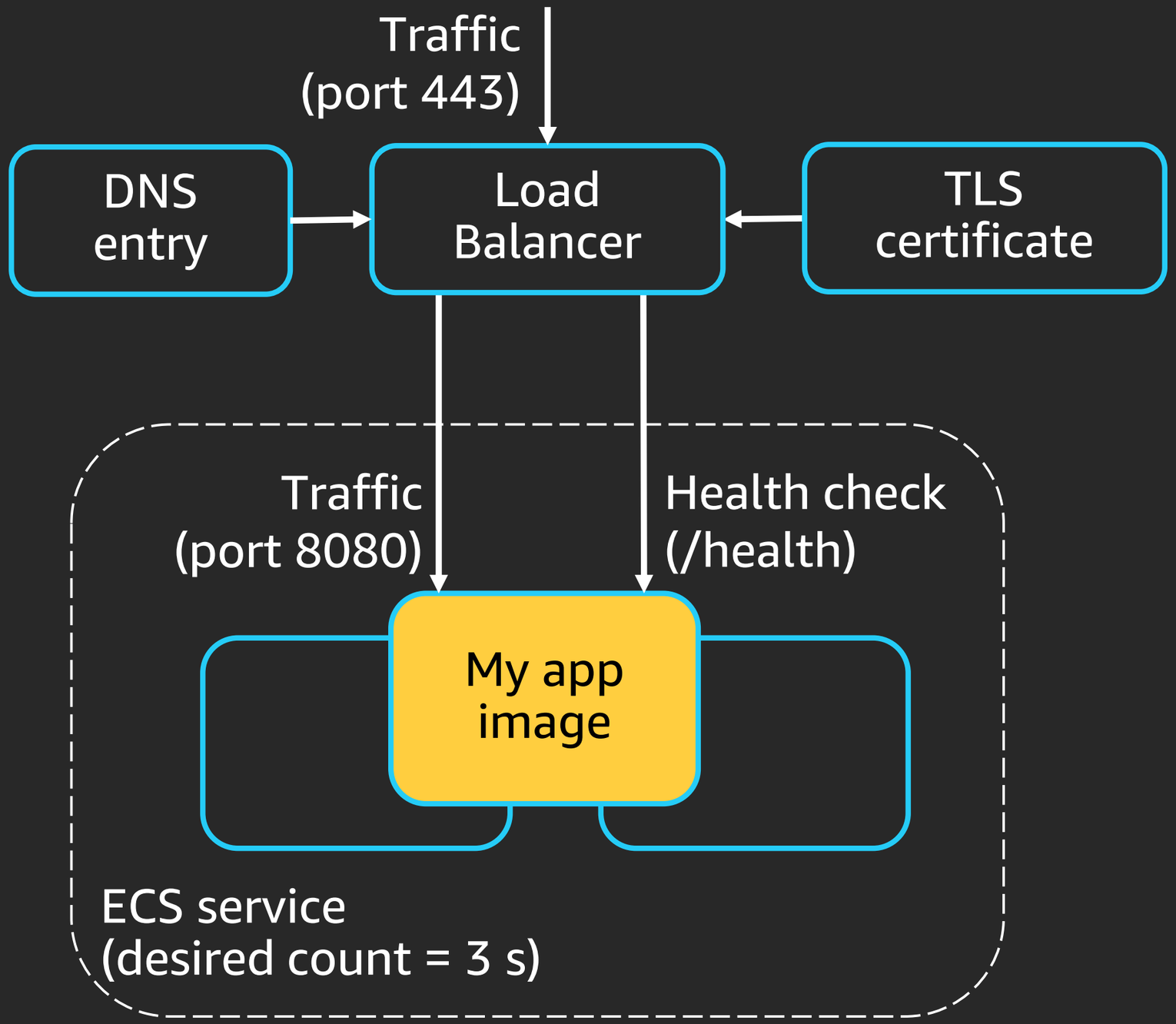
1. Automated releases

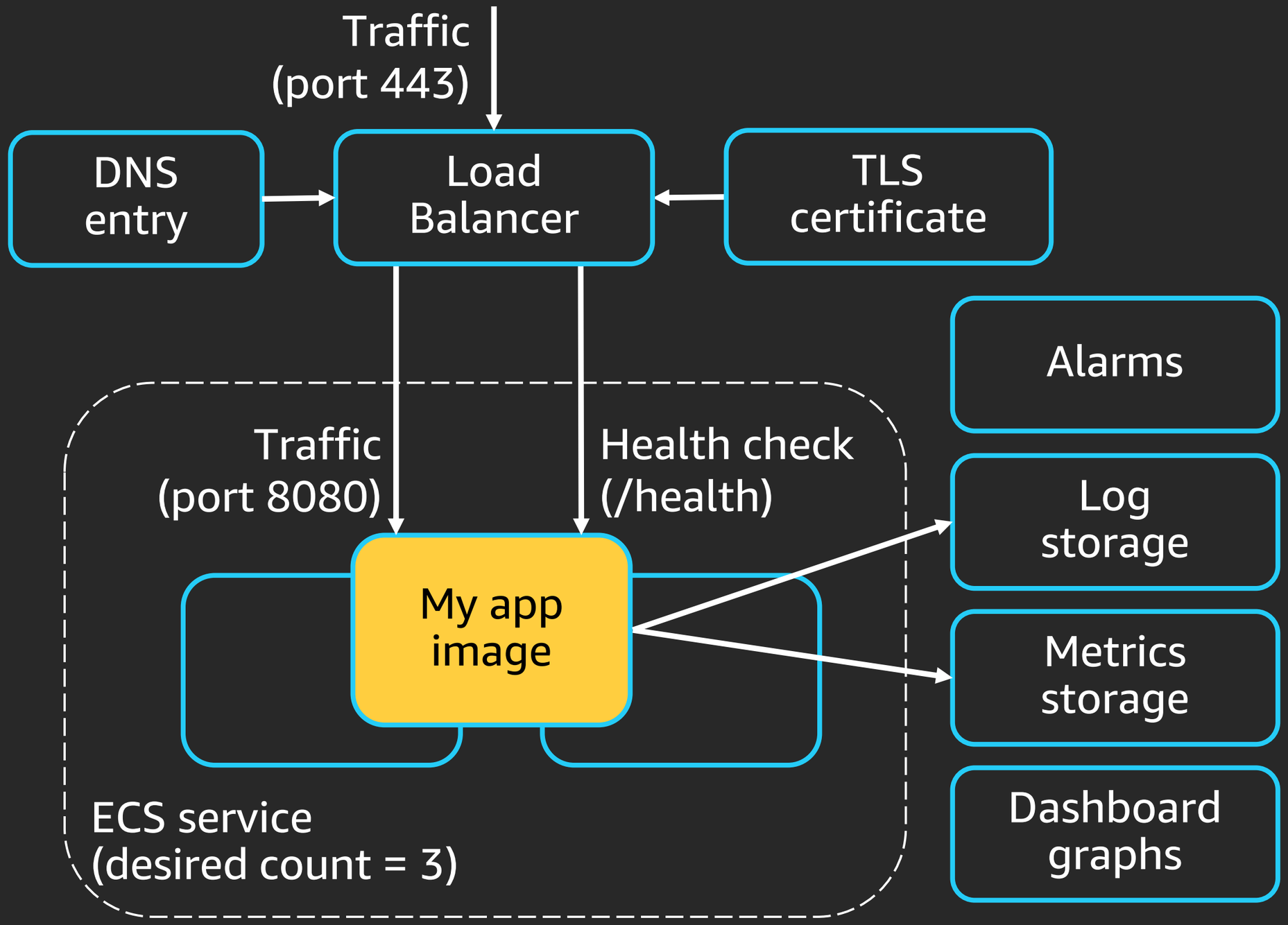
2. Safe deployments

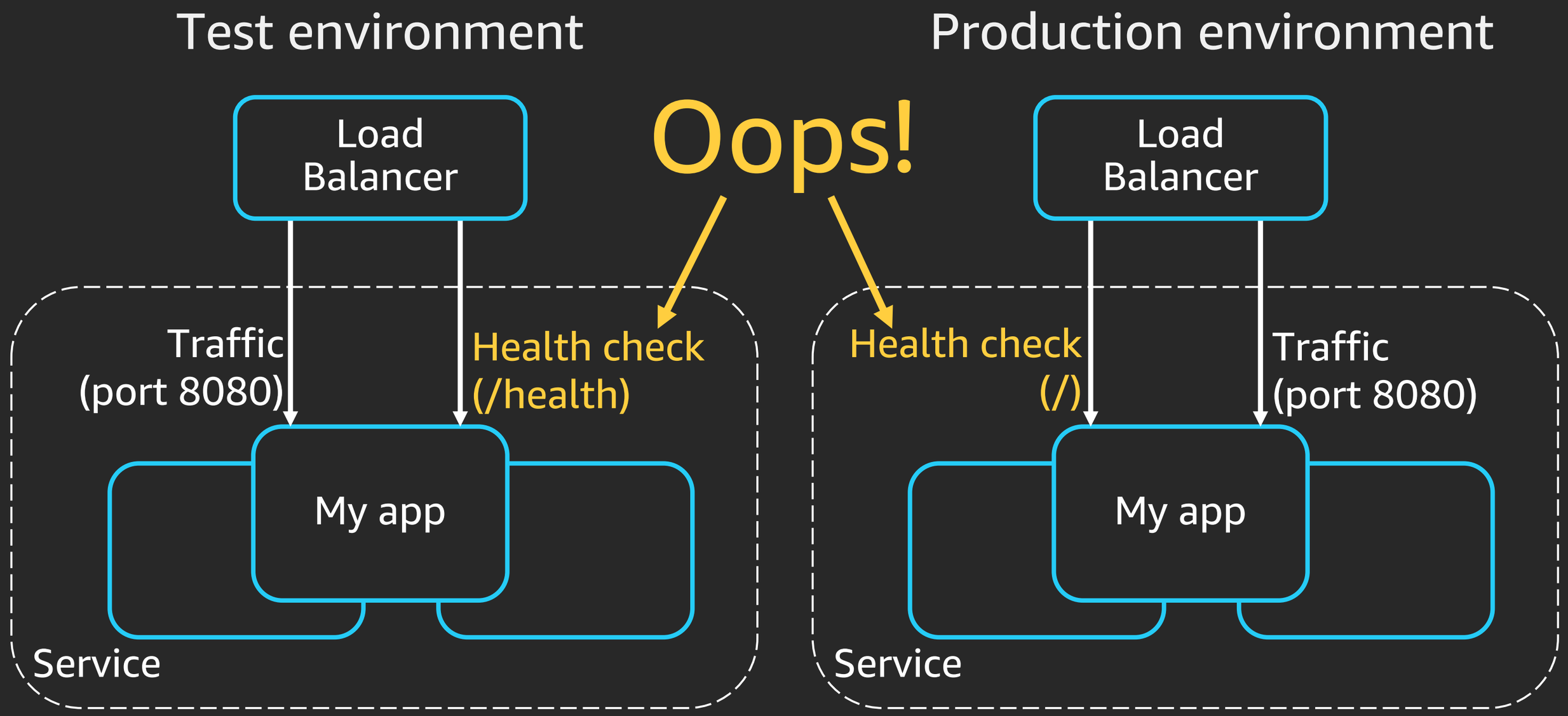
3. Repeatable infrastructure changes

My app image


My app
image
ECS service
(desired count = 3)


Traffic
(port 443)
DNS
entry
Load
Balancer
TLS
certificate
Traffic
(port 8080)
Health check
(/health)
My app
image
ECS service
(desired count = 3 s)

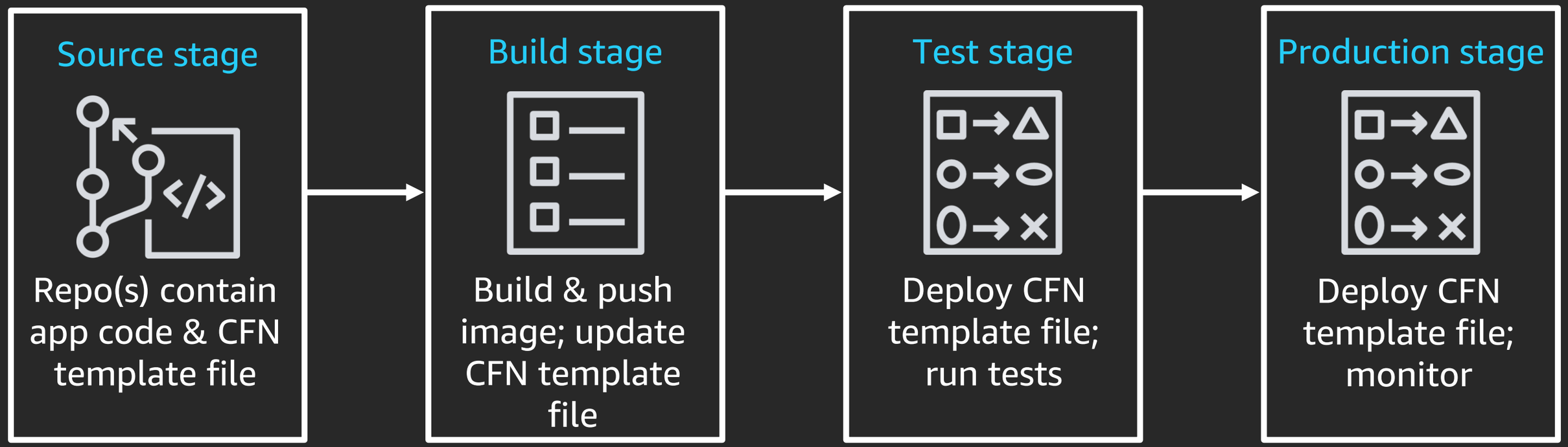

Traffic
(port 443)
DNS
entry
Load
Balancer
TLS
certificate
Alarms
Traffic
(port 8080)
Health check
(/health)
Log
storage
My app
image
Metrics
storage
Dashboard
graphs
ECS service
(desired count = 3)


Test environment
Production environment
Oops!
Load Balancer
Load Balancer
Traffic (port 8080)
Health check (/health)
Health check (/)
Traffic (port 8080)
My app
My app
Service
Service

Infrastructure as code with AWS CloudFormation

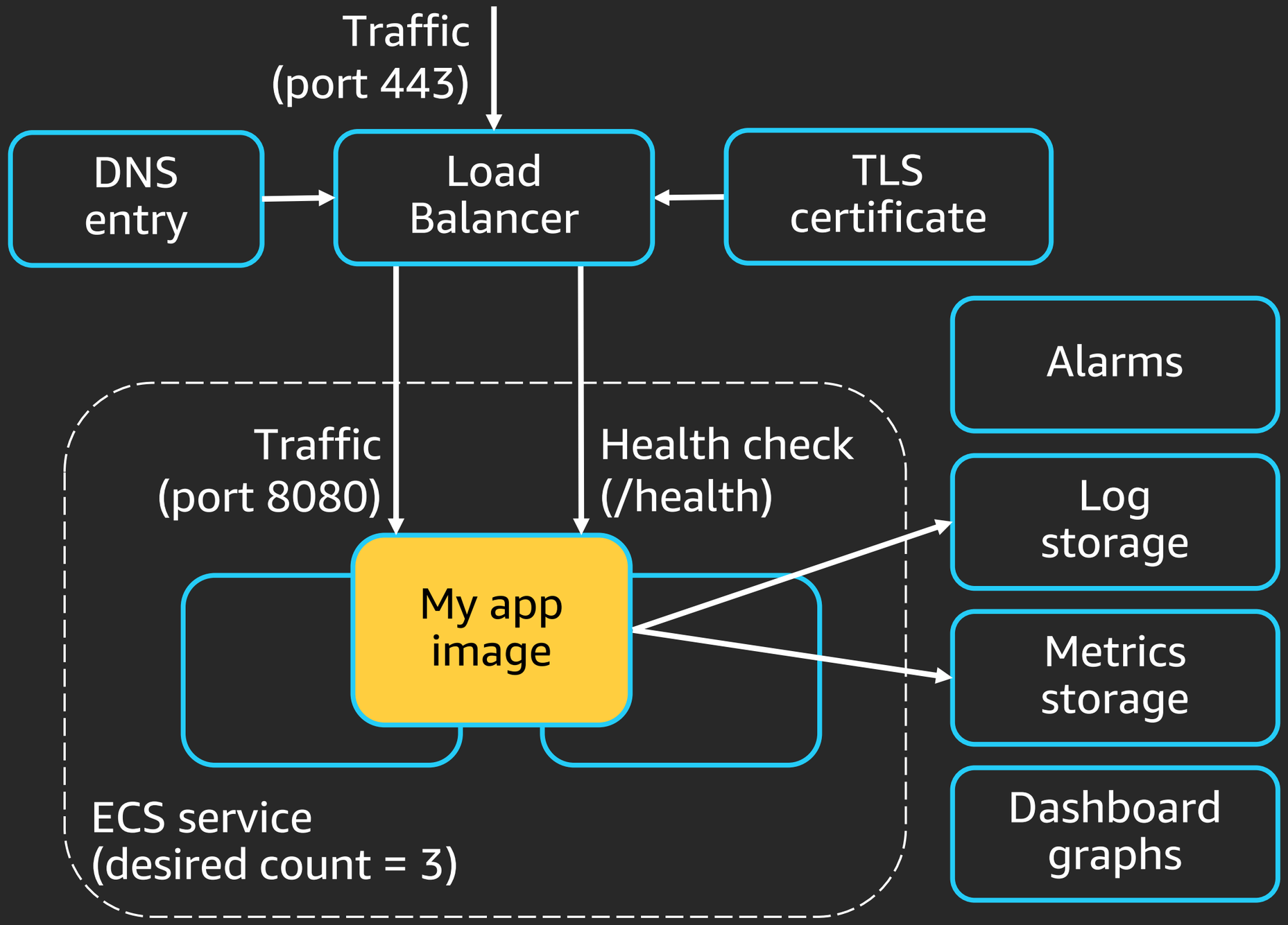
```
TargetGroup:
   Type: AWS::ElasticLoadBalancingV2::TargetGroup
   Properties:
     HealthCheckIntervalSeconds: 6
     HealthCheckPath: /health
     HealthCheckProtocol: HTTP
     HealthCheckTimeoutSeconds: 5
     HealthyThresholdCount: 2
     TargetType: ip
     Name: !Ref 'ServiceName'
     Port: !Ref 'ContainerPort'
     Protocol: HTTP
     UnhealthyThresholdCount: 2
```

Amazon's use of CI/CD and infrastructure as code



Insert image ID into infrastructure as code template

```
TaskDef:
  Type: AWS::ECS::TaskDefinition
  Properties:
    ContainerDefinitions:
-     - Image: my-app-image:<PLACEHOLDER>
+     - Image: my-app-image@sha256:e3b0c44298fc1c149afbf4c87...
```


Traffic
(port 443)
DNS
entry
Load
Balancer
TLS
certificate
Alarms
Traffic
(port 8080)
Health check
(/health)
Log
storage
My app
image
Metrics
storage
Dashboard
graphs
ECS service
(desired count = 3)

AWS Cloud Development Kit (AWS CDK)

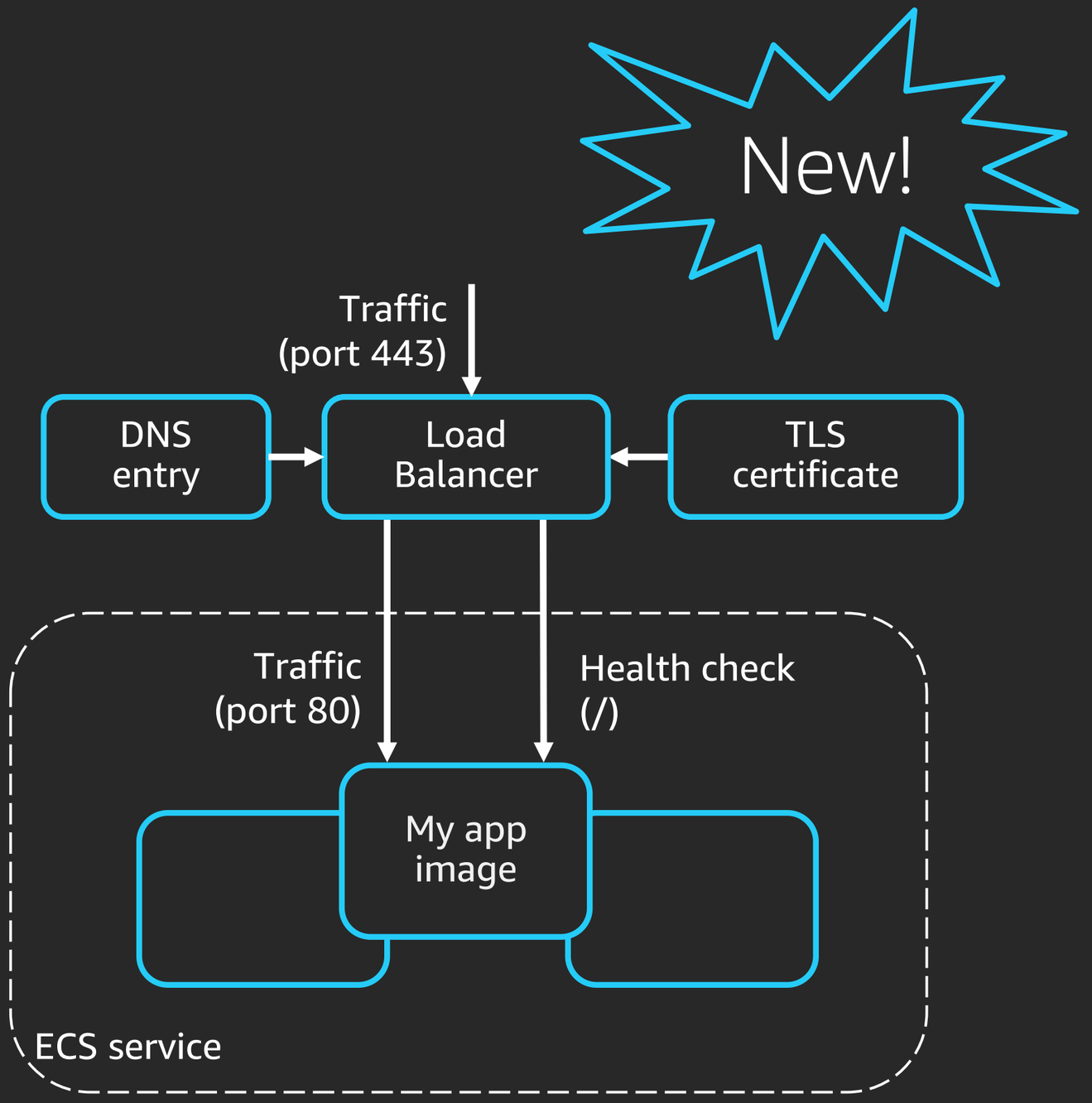
- Open-source framework to define cloud infrastructure in familiar programming languages such as TypeScript, Python, Java, and .NET
- Provisions resources with AWS CloudFormation
- Supports all AWS CloudFormation resource types
- Provides library of higher-level resource types that have AWS best practices built in by default

AWS CDK ECS constructs

```
const cluster = new ecs.Cluster(this, 'Cluster');

const taskDef = new ecs.FargateTaskDefinition(this, "MyTaskDefinition", {
  memoryLimitMiB: 512,
  cpu: 256,
});

taskDef.addContainer("AppContainer", {
  image: ecs.ContainerImage.fromRegistry("amazon/amazon-ecs-sample"),
});

new ecs.FargateService(this, "FargateService", {
  cluster,
  taskDefinition: taskDef
});
```

AWS CDK ECS patterns

- Application load balanced service
- Network load balanced service
- Queue processing service
- Scheduled task (Cron job)



AWS CDK ECS patterns

```
import { ApplicationLoadBalancedFargateService } from '@aws-cdk/aws-ecs-patterns';
import { ContainerImage } from 'aws-cdk/aws-ecs';
import cdk = require('@aws-cdk/core');

class BonjourFargate extends cdk.Stack {
  constructor(parent: cdk.App, name: string, props: cdk.StackProps) {
    super(parent, name, props);

    new ApplicationLoadBalancedFargateService(this, 'Service', {
      taskImageOptions: { image: ContainerImage.fromAsset('../src') }
    });
  }
}

const app = new cdk.App();
new BonjourFargate(app, 'Bonjour', {});
app.synth();
```

ECS pattern includes:

- VPC
- ECS cluster
- ECS task definition
- AWS Fargate service
- Amazon CloudWatch Logs
- Load Balancer
- Security groups

AWS CDK ECS patterns

```
import { ApplicationLoadBalancedFargateService } from '@aws-cdk/aws-ecs-patterns';
import { ContainerImage } from 'aws-cdk/aws-ecs';
import cdk = require('@aws-cdk/core');

class BonjourFargate extends cdk.Stack {
  constructor(parent: cdk.App, name: string, props: cdk.StackProps) {
    super(parent, name, props);

    new ApplicationLoadBalancedFargateService(this, 'Service', {
      taskImageOptions: { image: ContainerImage.fromAsset('../src') }
    });
  }
}

const app = new cdk.App();
new BonjourFargate(app, 'Bonjour', {});
app.synth();
```

AWS CDK will create an ECR repository, then build and push your Docker image

AWS CDK ECS patterns

```
import { ApplicationLoadBalancedFargateService } from '@aws-cdk/aws-ecs-patterns';
import { ContainerImage } from 'aws-cdk/aws-ecs';
import cdk = require('@aws-cdk/core');

class BonjourFargate extends cdk.Stack {
  constructor(parent: cdk.App, name: string, props: cdk.StackProps) {
    super(parent, name, props);

    new ApplicationLoadBalancedFargateService(this, 'Service', {
      taskImageOptions: { image: ContainerImage.fromAsset('../src') }
    });
  }
}

const app = new cdk.App();
new BonjourFargate(app, 'Bonjour', {});
app.synth();
```

Generates hundreds of lines of AWS CloudFormation template

AWS CDK CLI

```
npm install -g aws-cdk

cdk init --language typescript

cdk synth

cdk deploy
```

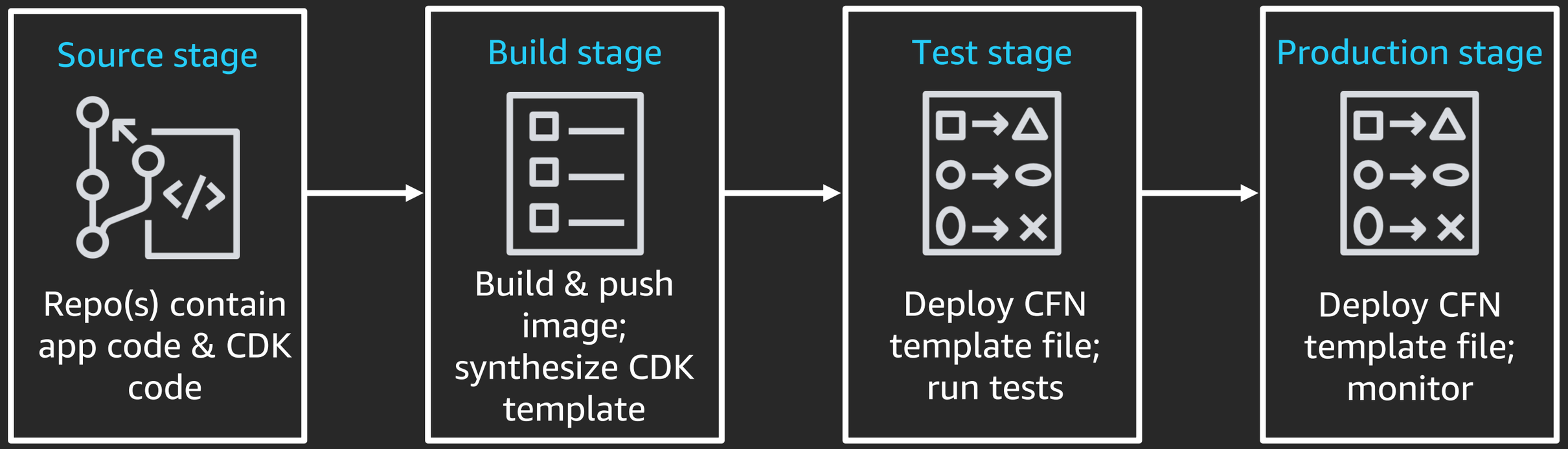
CodePipeline

Use AWS CloudFormation deployment actions with synthesized CDK application

Jenkins

Use AWS CDK CLI to deploy

CI/CD for AWS CDK code



AWS CDK CodeBuild buildspec

```
version: 0.2
phases:
  install:
    commands:
      # Install CDK
      - npm install -g aws-cdk
  build:
    commands:
      # Compile code & synthesize CloudFormation templates
      - npm ci
      - tsc
      - cdk synth --app 'node ecs-service.js'
```

AWS CDK CodePipeline pipelines

```
class DeployCDKPipeline extends cdk.Stack {
    constructor(parent: cdk.App, name: string, props: MyMicroservicePipelineProps) {
        super(parent, name, props);

        const pipeline = new codepipeline.Pipeline(this, 'Pipeline', {
            pipelineName: props.serviceName,
        });

        const githubAccessToken = cdk.SecretValue.secretsManager('GitHubToken');
        const sourceOutput = new codepipeline.Artifact('SourceArtifact');
        const sourceAction = new actions.GitHubSourceAction({
            actionName: 'GitHubSource', output: sourceOutput,
            owner: ‘my-github-org’, repo: props.serviceName,
            oauthToken: githubAccessToken
       });
```

AWS CDK CodePipeline pipelines

```
class MyMicroservicePipelinesStack extends cdk.Stack {
    constructor(parent: cdk.App, name: string, props?: cdk.StackProps) {
        super(parent, name, props);

        new DeployCDKPipeline(this, 'Pipeline1', { 'serviceName': 'Microservice1' });
        new DeployCDKPipeline(this, 'Pipeline2', { 'serviceName': 'Microservice2' });
        new DeployCDKPipeline(this, 'Pipeline3', { 'serviceName': 'Microservice3' });
        new DeployCDKPipeline(this, 'Pipeline4', { 'serviceName': 'Microservice4' });
    }
}
```

Best practices for CI/CD

1. Automated releases

2. Safe deployments

3. Repeatable infrastructure changes

Best practices for CI/CD

1. Automated releases

2. Safe deployments

3. Repeatable infrastructure changes

Demo

Best practices for CI/CD

1. Automated releases

2. Safe deployments

3. Repeatable infrastructure changes

Learn more

Related breakouts:

- DEV209: Introduction to DevOps on AWS
- SVS336: CI/CD for serverless applications

Demo code:

https://github.com/aws-samples/aws-reinvent-2019-trivia-game

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.

AWS re:Invent

aws