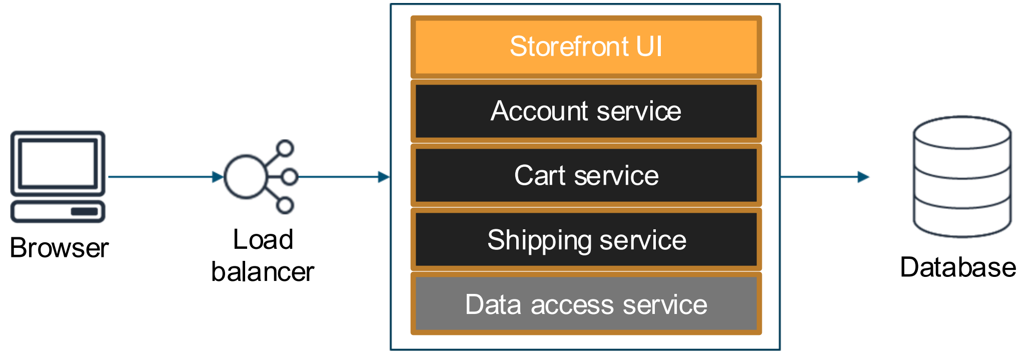
Cloud Services 2

Amazon Web Services

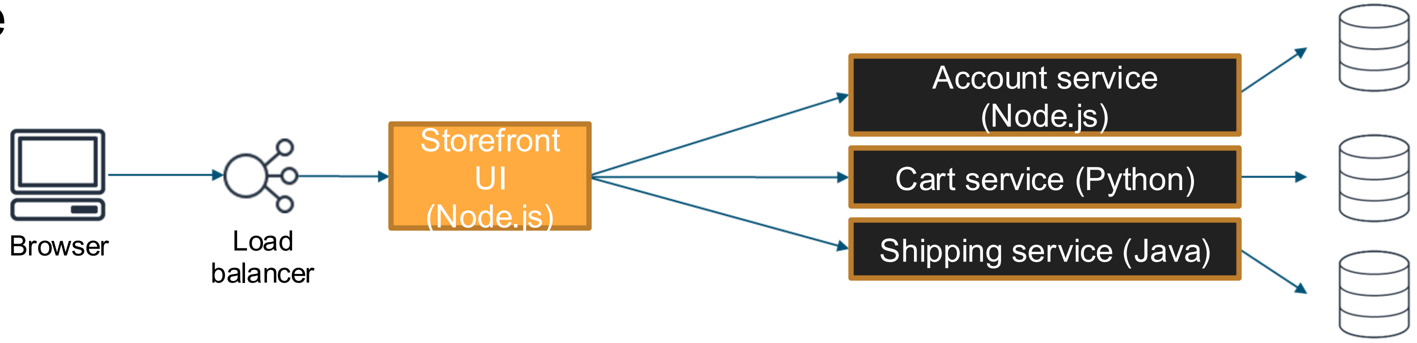
AWS Container Services

Comparing monolithic and microservice architectures

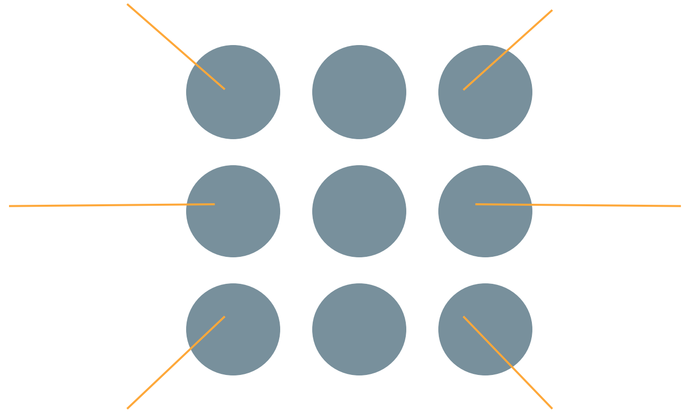
Monolithic



Microservices



Characteristics of microservices



Decentralized

Polyglot

Independent

Specialized

Black boxes

You build it, you run it

Microservices and containers

Microservices design	Container characteristics
• Decentralized, evolutionary design • Smart endpoints, dumb pipes	• Each container uses the language and technology that are best suited for the service. • Each component or system in the architecture can be isolated, and can evolve separately, instead of updating the system in a monolithic style.
• Independent products, not projects	• You can use containers to package all of your dependencies and libraries into a single, immutable object.
• Designed for failure • Disposable	• You can gracefully shut down a container when something goes wrong and create a new instance. You start fast, fail fast, and release any file handlers. • The development pattern is like a circuit breaker. Containers are added and removed, workloads change, and resources are temporary because they constantly change.
• Development and production parity	• Containers can make development, testing, and production environments consistent. • This consistency facilitates DevOps, in which a containerized application that works on a developer's system will work the same way on a production system.

Challenges of managing containers at scale

- State of containers
- Scheduling of starts and stops
- Resources available on each server
- Maximizing availability, resilience, and performance

Container orchestration platforms

Scheduling

Placement

Service integration

Amazon ECS

Amazon Elastic Container Service (Amazon ECS)

Fully managed container orchestration service

- Scales rapidly to thousands of containers with no additional complexity
- Schedules placement across managed clusters
- Integrates with third-party schedulers and other AWS services

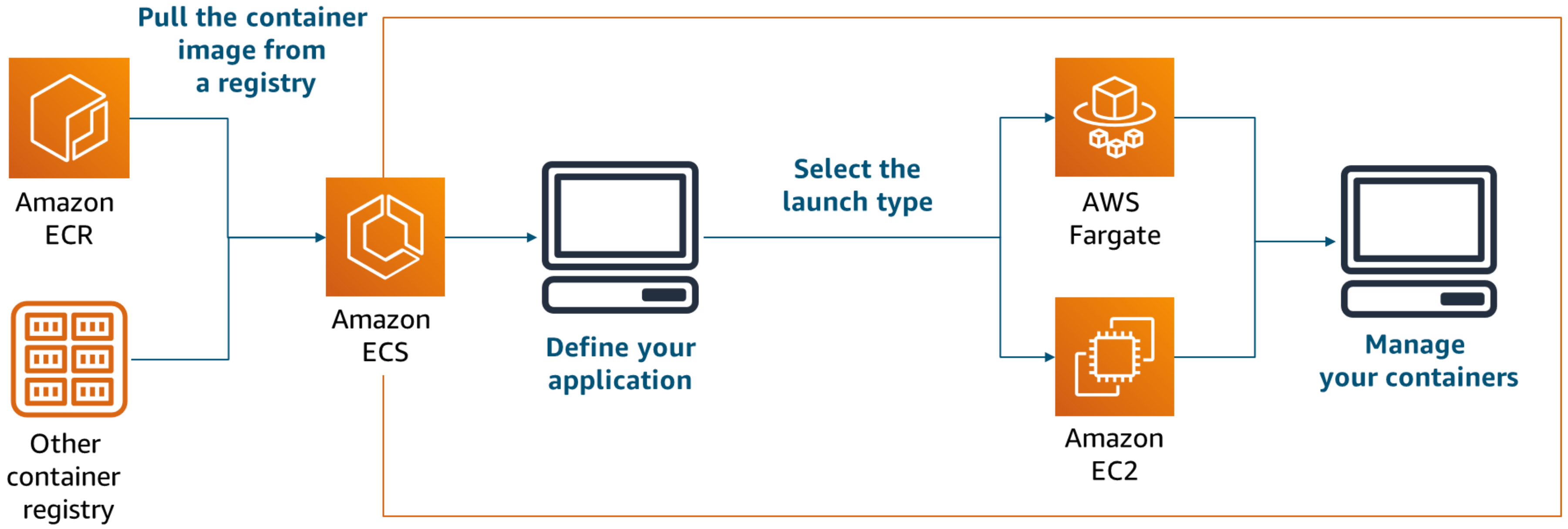
Amazon ECR

Amazon Elastic Container Registry (Amazon ECR)

Fully managed container registry that you can use to easily store, run, and manage container images for applications that run on Amazon ECS

- Scalable and highly available
- Integrated with Amazon ECS and Docker CLI
- Secure:
  - Encryption at rest
  - Integration with the AWS Identity and Access Management Service (IAM)

Amazon ECS solution architecture

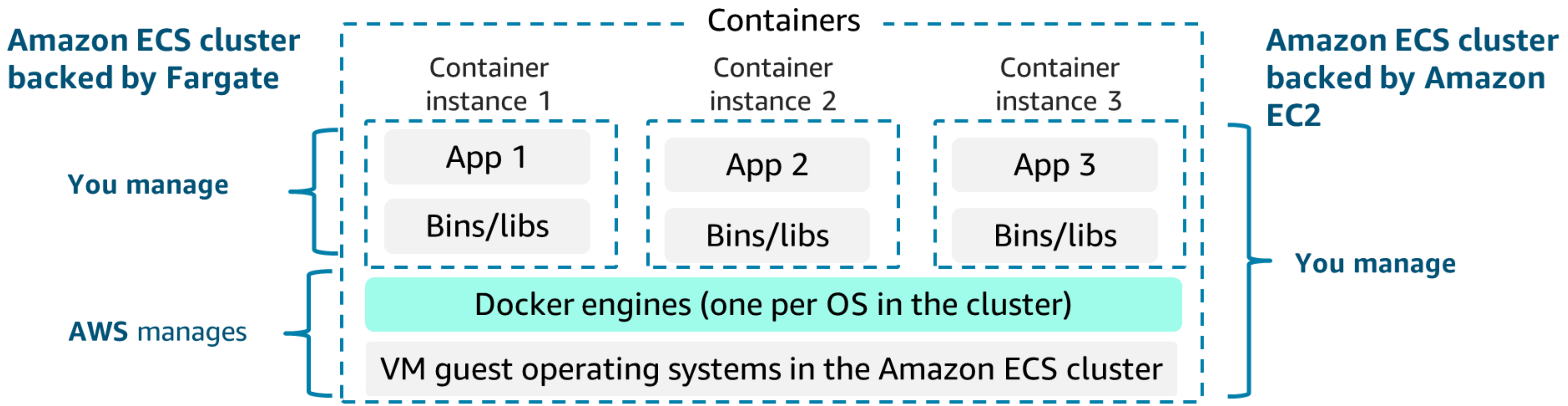


AWS Fargate

AWS Fargate

- Is a fully managed container service
- Works with Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS)
- Provisions, manages, and scales your container clusters
- Manages runtime environment
- Provides automatic scaling

Amazon ECS with Fargate or Amazon EC2


Amazon ECS cluster backed by Fargate
You manage
AWS manages
Containers
Container instance 1
Container instance 2
Container instance 3
App 1
Bins/libs
App 2
Bins/libs
App 3
Bins/libs
Docker engines (one per OS in the cluster)
VM guest operating systems in the Amazon ECS cluster
Amazon ECS cluster backed by Amazon EC2
You manage

Choose Fargate:

Services subject to wide swings in demand

Large workloads that are optimized for low overhead

Small test environments

Batch workloads that run on a schedule

Choose Amazon EC2:

More predictable resource requirements, or the option of using reserved instances to reduce costs

Large workloads that are optimized for price

Compliance with organizational security requirements

Excess Amazon EC2 capacity

Creating an Amazon ECR repository and pushing an image

```
# Create a repository called hello-world
> aws ecr create-repository \
    --repository-name hello-world \
    --region us-east-1

# Build and tag an image
> docker build -t hello-world .
> docker tag hello-world:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest

# Authenticate Docker to your Amazon ECR registry
# You can skip the `docker login` step if you have amazon-ecr-credential-helper set up
> aws ecr get-login-password --region region | docker login --username AWS --password-stdin
aws_account_id.dkr.ecr.region.amazonaws.com

# Push an image to your repository
> docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

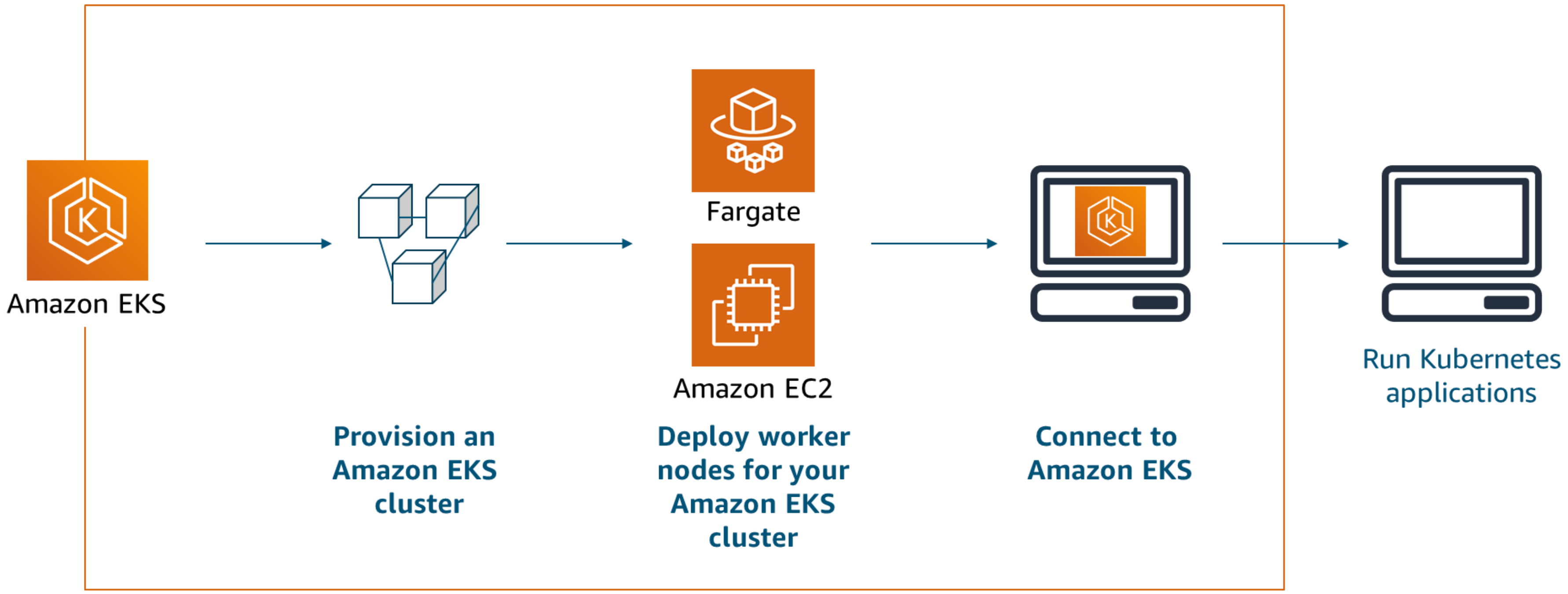
Amazon EKS

Amazon Elastic Kubernetes Service (Amazon EKS)

Managed service that runs Kubernetes on the AWS Cloud

- Built with the Kubernetes community
- Conformant and compatible
- Secure by default

Amazon EKS


Amazon EKS
Fargate
Amazon EC2
Provision an Amazon EKS cluster
Deploy worker nodes for your Amazon EKS cluster
Connect to Amazon EKS
Run Kubernetes applications

Elastic Beanstalk

AWS Elastic Beanstalk

Service for deploying and scaling web applications and services

- Automatically handles deployment details like capacity provisioning, load balancing, automatic scaling, and application health monitoring
- Provides a variety of platforms on which to build your applications
- Use to manage all of the resources that run your application as an environment

Elastic Beanstalk components

Command	Description
Application	Logical collection of Elastic Beanstalk components. Conceptually similar to a folder.
Application version	Specific, labeled iteration of deployable code for a web application.
Environment	Collection of AWS resources that run an application version.
Environment tier	Designation of the type of application that the environment runs. Determines what resources Elastic Beanstalk provisions to support it.
Environment configuration	Collection of parameters and settings that define how an environment and its associated resources behave.
Saved configuration	Template that you can use as a starting point for creating unique environment configurations.
Platform	Combination of an OS, programming language runtime, web server, application server, and Elastic Beanstalk components. You design and target your web application to a platform.
Elastic Beanstalk CLI	CLI for Elastic Beanstalk. Provides interactive commands that simplify creating, updating, and monitoring environments from a local repository.

IAM permissions in Elastic Beanstalk environments

IAM roles assigned during environment creation

Service Role

Assigned during creation

Elastic Beanstalk assumes that it uses other services on your behalf

Default service role:

```
aws-elasticbeanstalk-
service-role
```

Instance profile

Assigned during creation

Applied to instances that are launched in your environment

Default instance profile:

```
aws-elasticbeanstalk-
ec2-role
```

User policies

Optionally assigned

Can be **attached to users or groups** who create and manage Elastic Beanstalk applications and environments

Two managed user policies are available to grant either full administrative access or read-only access

AWS Managed policy example

```
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : "ec2:Describe*",
      "Resource" : "*"
    },
    {
      "Effect" : "Allow",
      "Action" :
"elasticloadbalancing:Describe*",
      "Resource" : "*"
    },
… }
    truncated for brevity
```

Excerpt of
AmazonEC2ReadOnlyAccess

Elastic Beanstalk simplifies container deployment

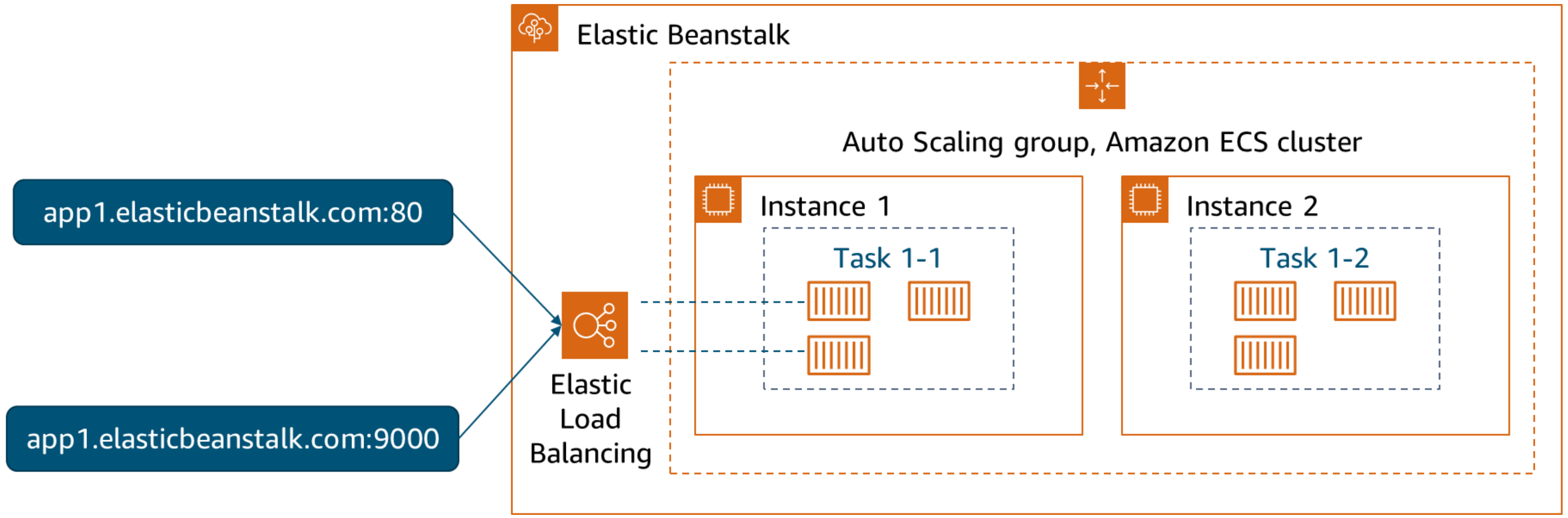
Getting started with
Amazon ECS

1. Create a task definition
2. Create and configure a cluster including:
   - EC2 instances
   - VPC settings
   - IAM role definition
3. Create a service to run and maintain a specified number of instances of a task

Getting started with
Elastic Beanstalk

1. Write a Dockerrun.aws.json file and provide your zipped code
2. Select the platform for your language
3. Launch your application

Multicontainer Docker platform


app1.elasticbeanstalk.com:80
app1.elasticbeanstalk.com:9000
Elastic Beanstalk
Auto Scaling group, Amazon ECS cluster
Instance 1
Task 1-1
Instance 2
Task 1-2
Elastic Load Balancing

Deployment option namespaces

`aws:elasticbeanstalk:command`

- Choose the deployment policy
- Set a timeout
- Choose options for size and type of batches to use
- Choose whether to cancel deployment on a failed health check

`aws:elasticbeanstalk:trafficsplitting`

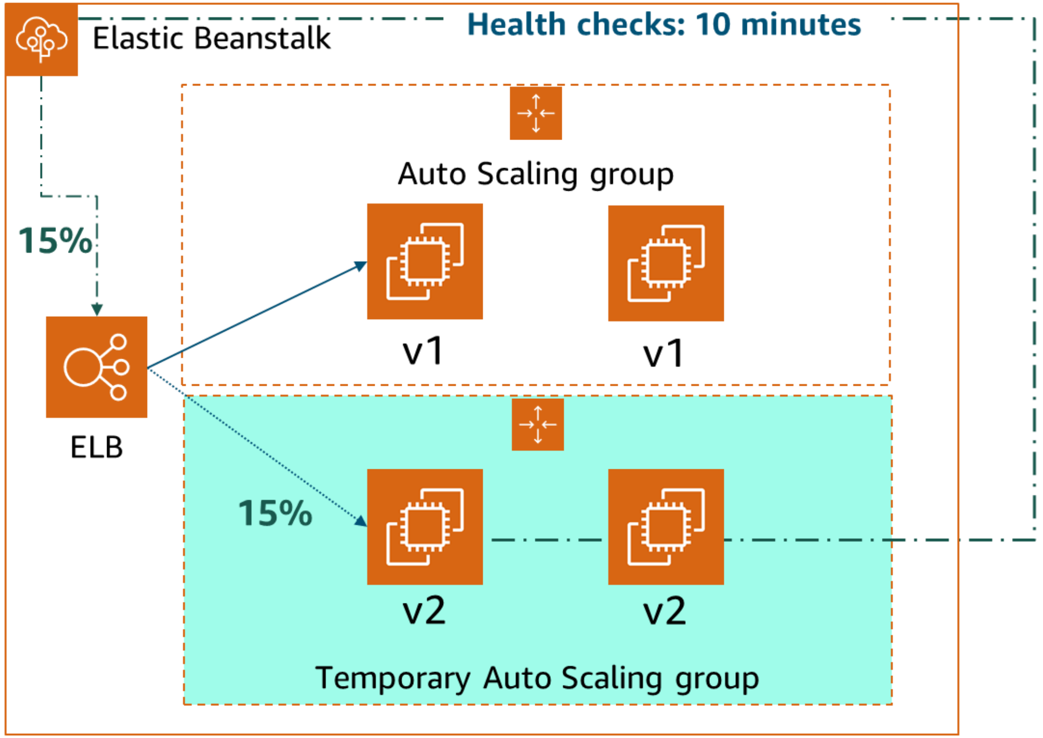
- Choose the percentage of traffic to go to new instances
- Choose how long to wait before continuing to shift more traffic

Example of traffic splitting (canary testing)

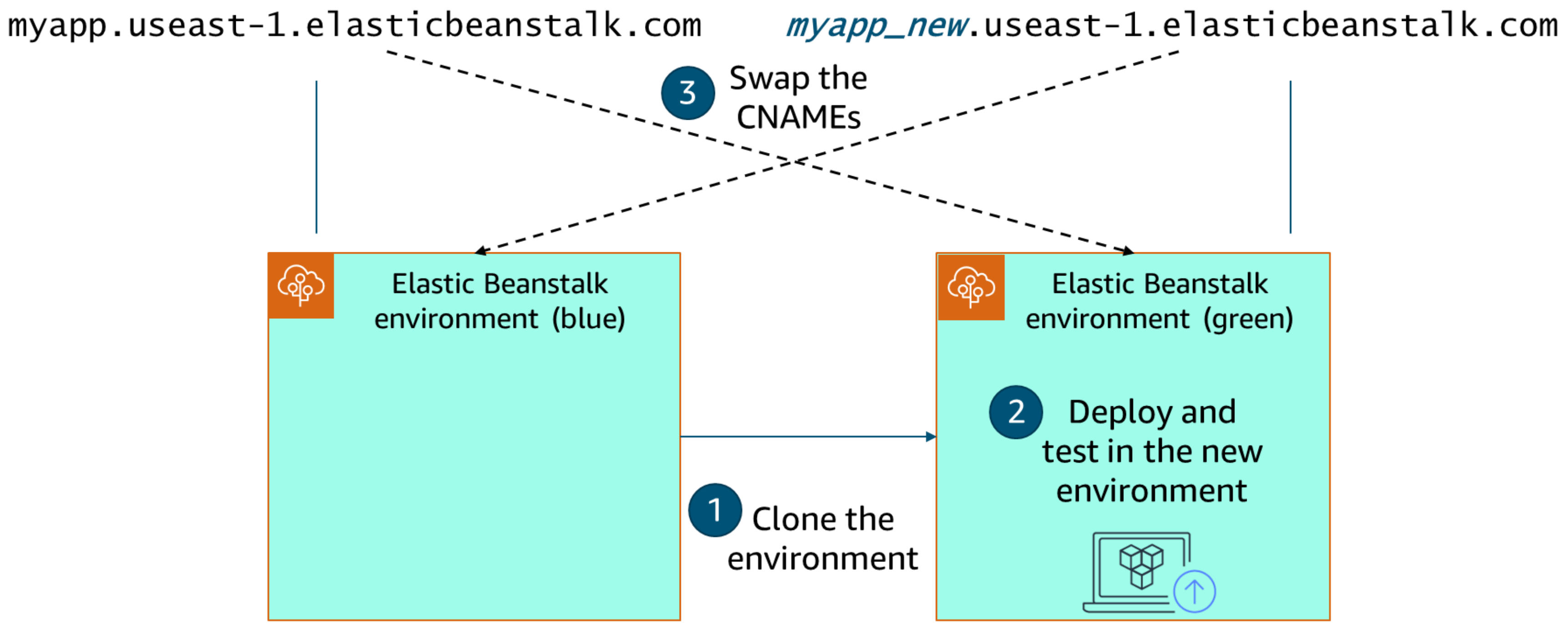
Example deployment configurations

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy:
TrafficSplitting

aws:elasticbeanstalk:trafficsplitting
:
    NewVersionPercent: "15"
    EvaluationTime: "10"
```



Blue/green deployments on Elastic Beanstalk


myapp.useast-1.elasticbeanstalk.com
myapp_new.useast-1.elasticbeanstalk.com
3 Swap the CNAMEs
Elastic Beanstalk environment (blue)
Elastic Beanstalk environment (green)
2 Deploy and test in the new environment
1 Clone the environment

Extra leermaterialen & labs

- Documentatie:
  - https://docs.aws.amazon.com/ecs/
  - https://docs.aws.amazon.com/ecr/