Errors or corrections? Email us at aws-course-feedback@amazon.com.
Other questions? Email us at aws-training-info@amazon.com.

Architecting with AWS

Elasticity, Scalability, and Bootstrapping

Elasticity, Scalability, and Bootstrapping | What we'll cover

1. Basic tenets of AWS
2. Patterns and (anti-patterns) for creating scalable architectures in AWS
3. Bootstrapping EC2 Instances
4. Building with CloudFormation
5. Components of Auto Scaling

Elasticity, Scalability, and Bootstrapping | What we'll cover

1

Basic Tenets of AWS

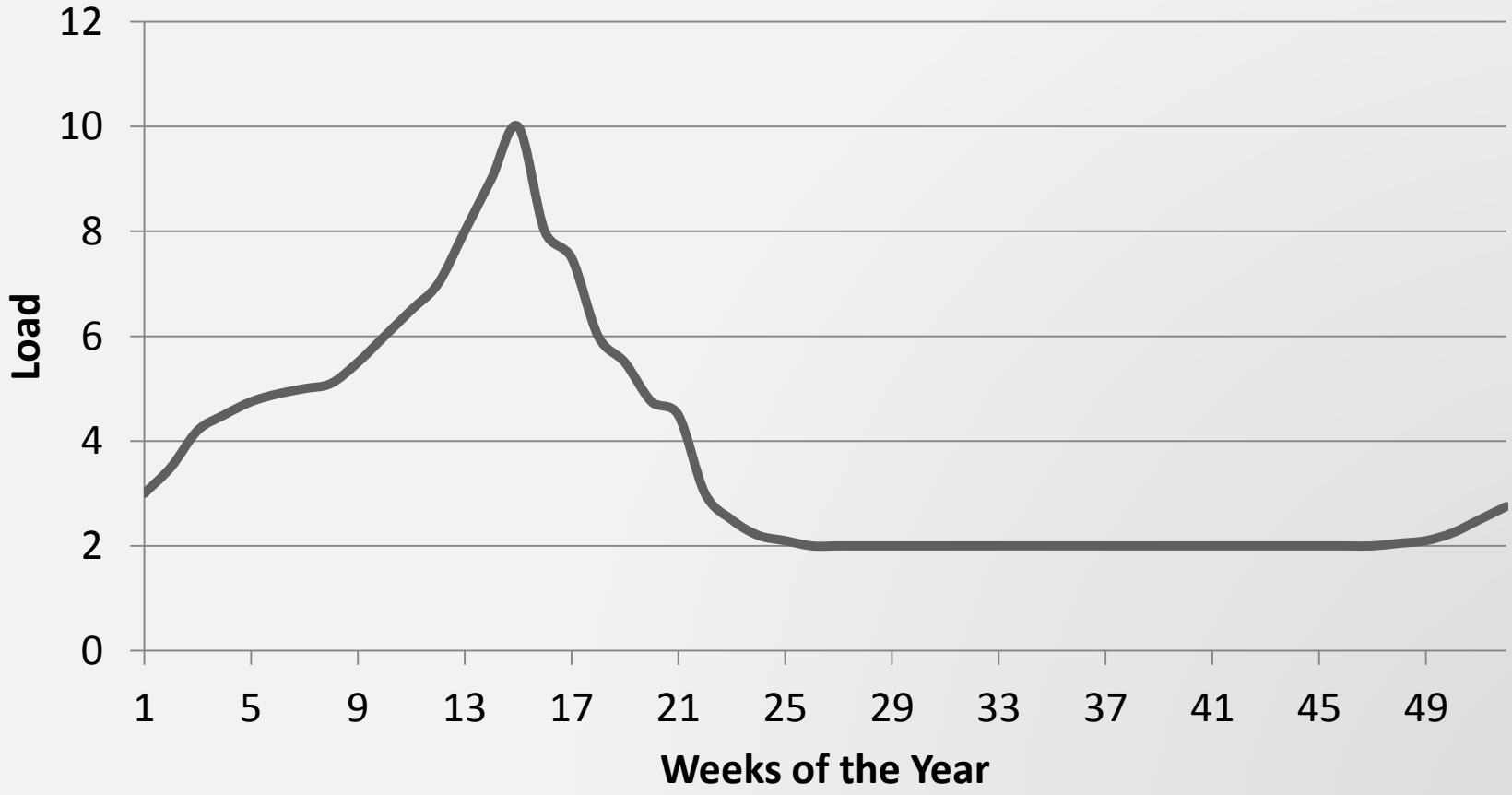
Objectives

1. Review how traditional architectures accommodate expected load variation
2. Anti-patterns for elastic, scalable architectures
3. 4 patterns for elastic, scalable architectures

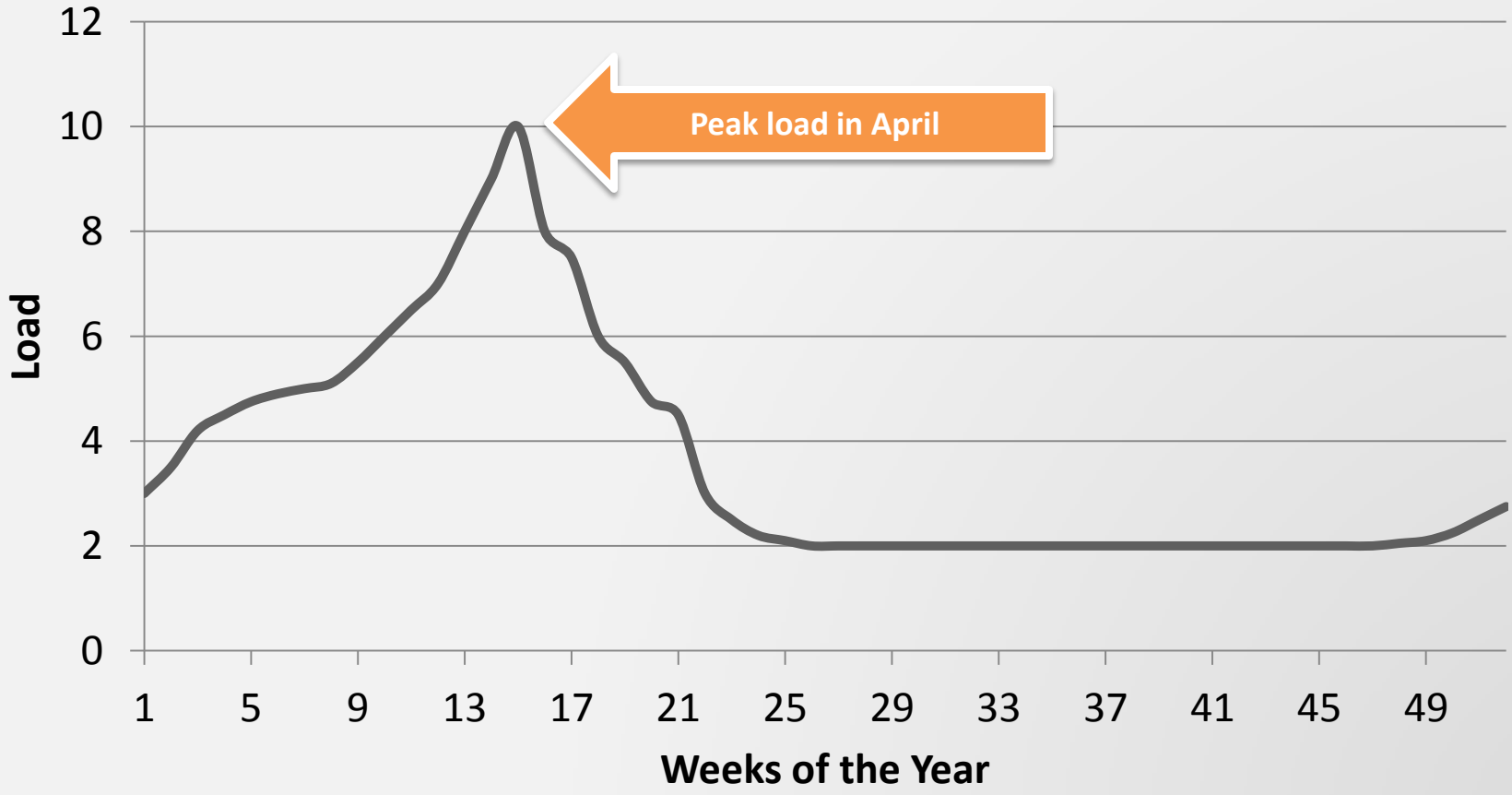
Expected Load Variation

- Non-cloud systems typically over-provision and under-utilize
- Under-utilization costs: capital, space, power, cooling, and maintenance

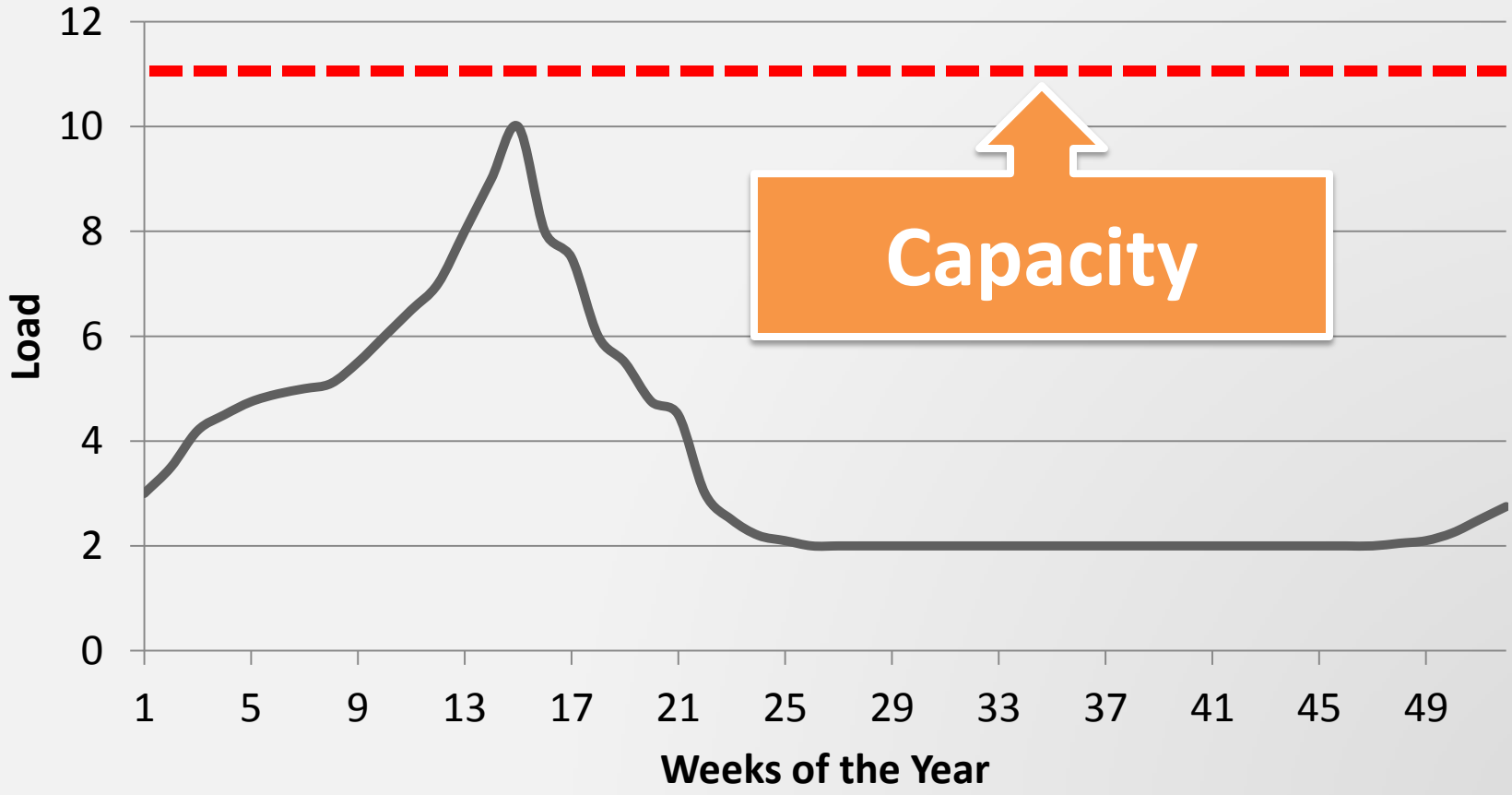
Expected Load Variation



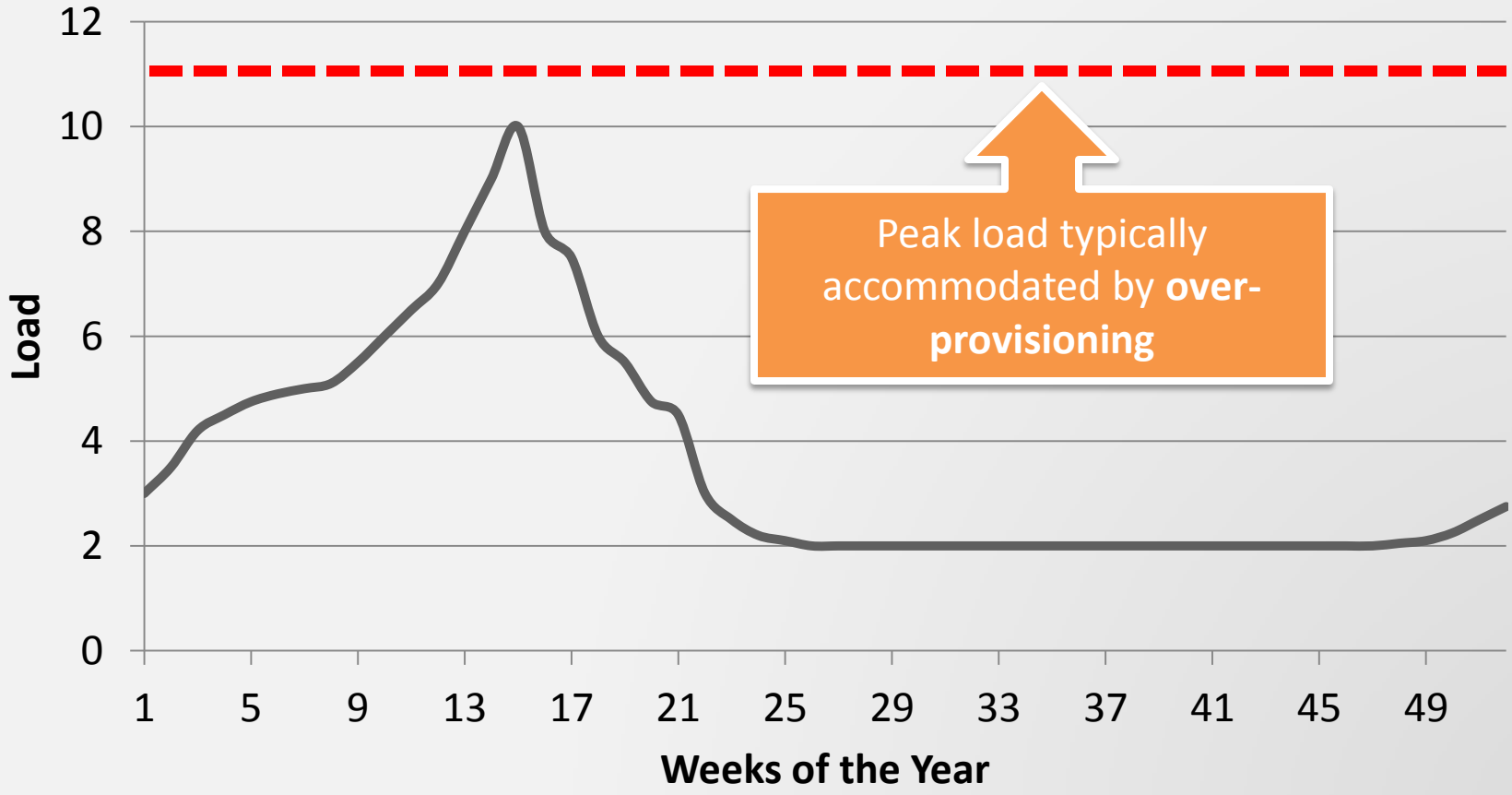
Expected Load Variation



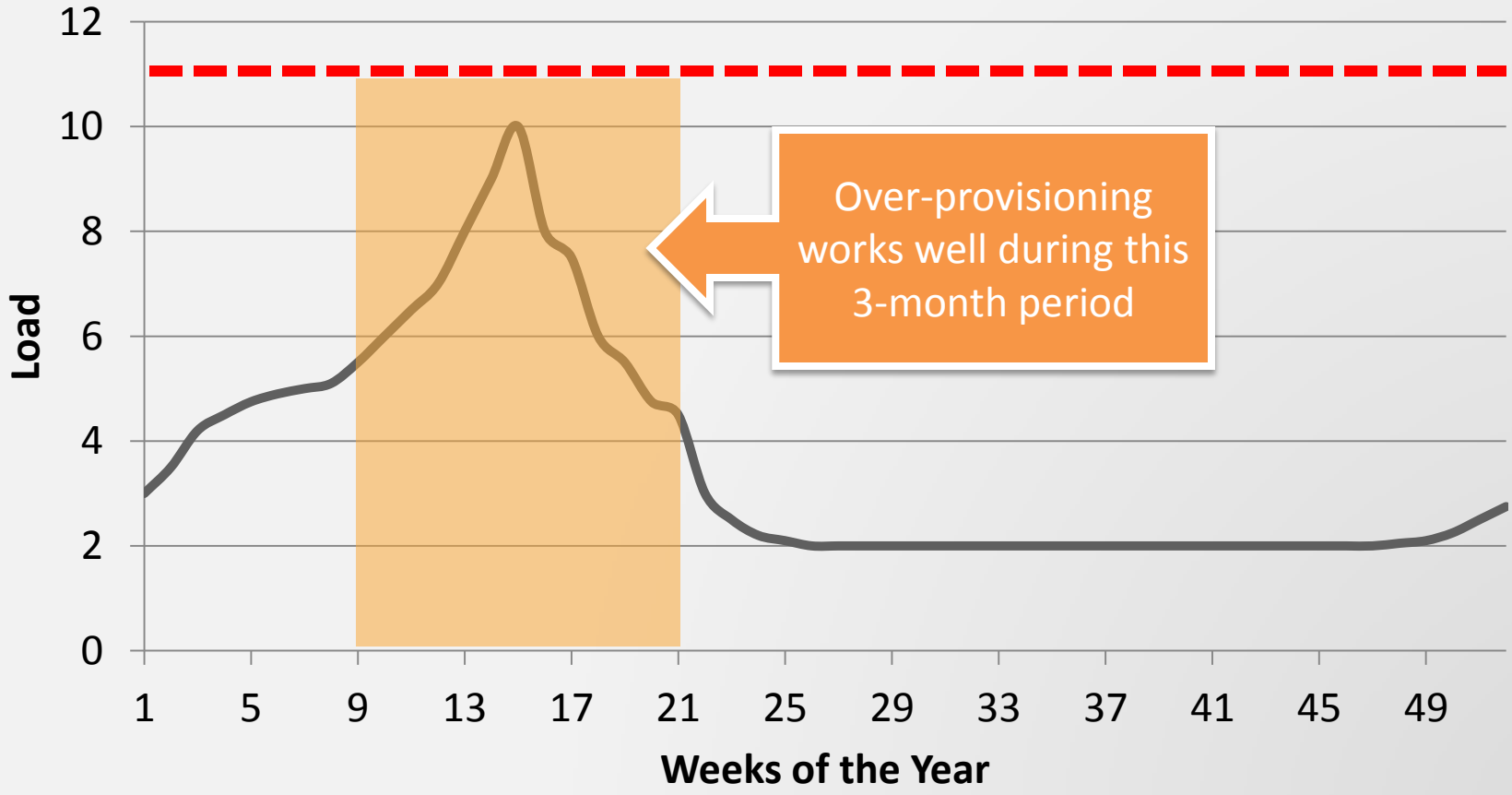
Expected Load Variation



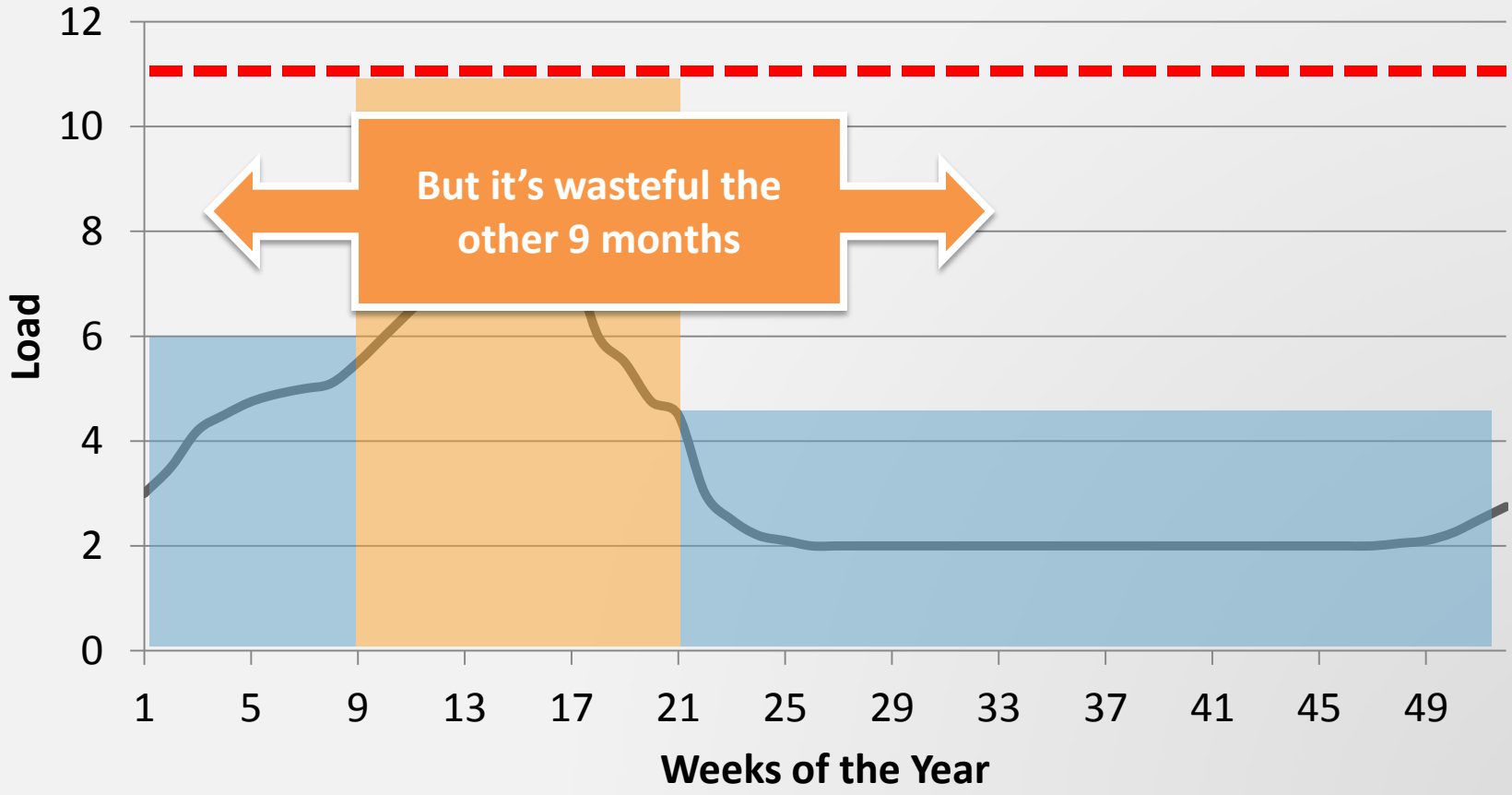
Expected Load Variation



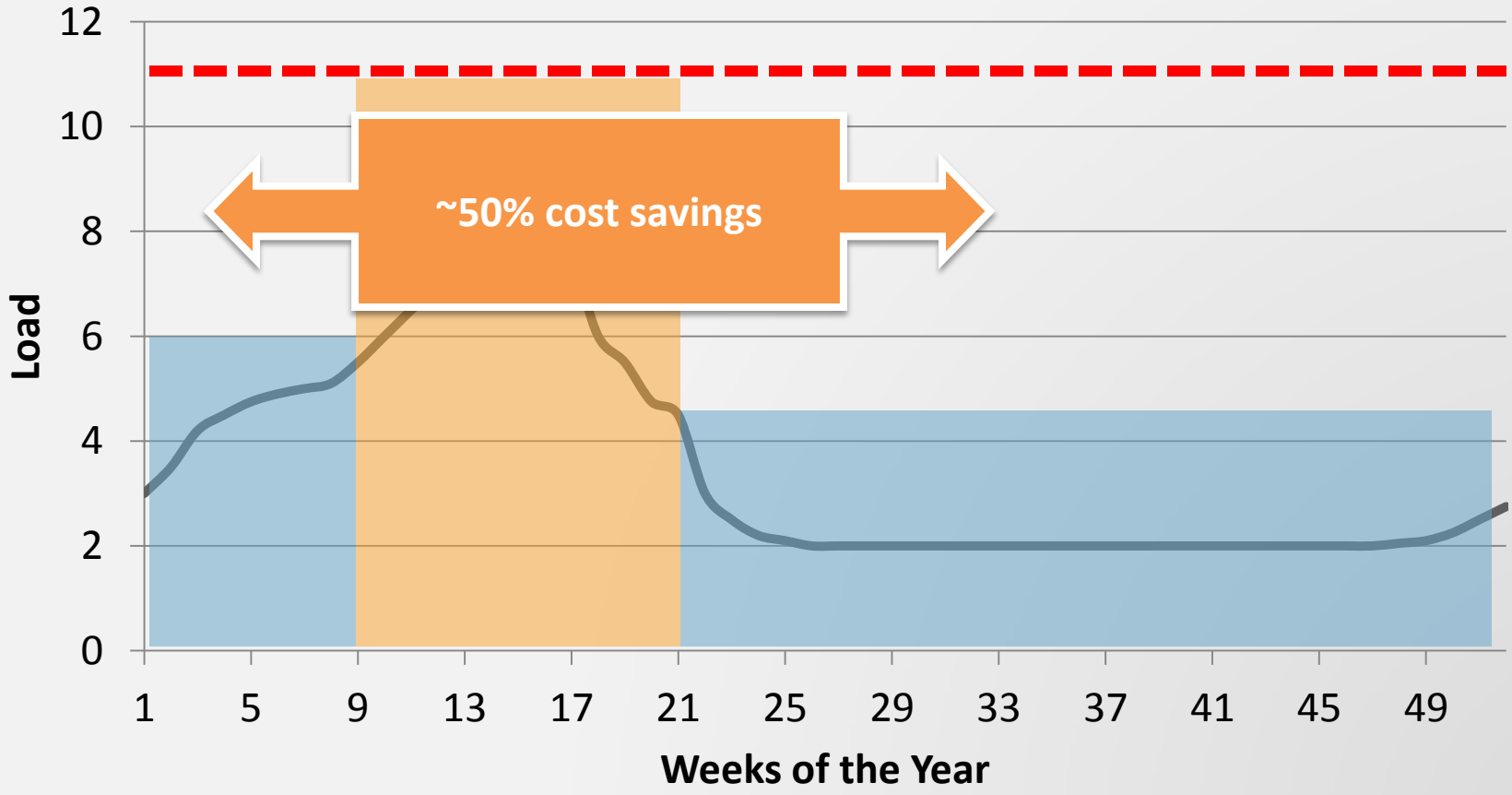
Expected Load Variation



Expected Load Variation



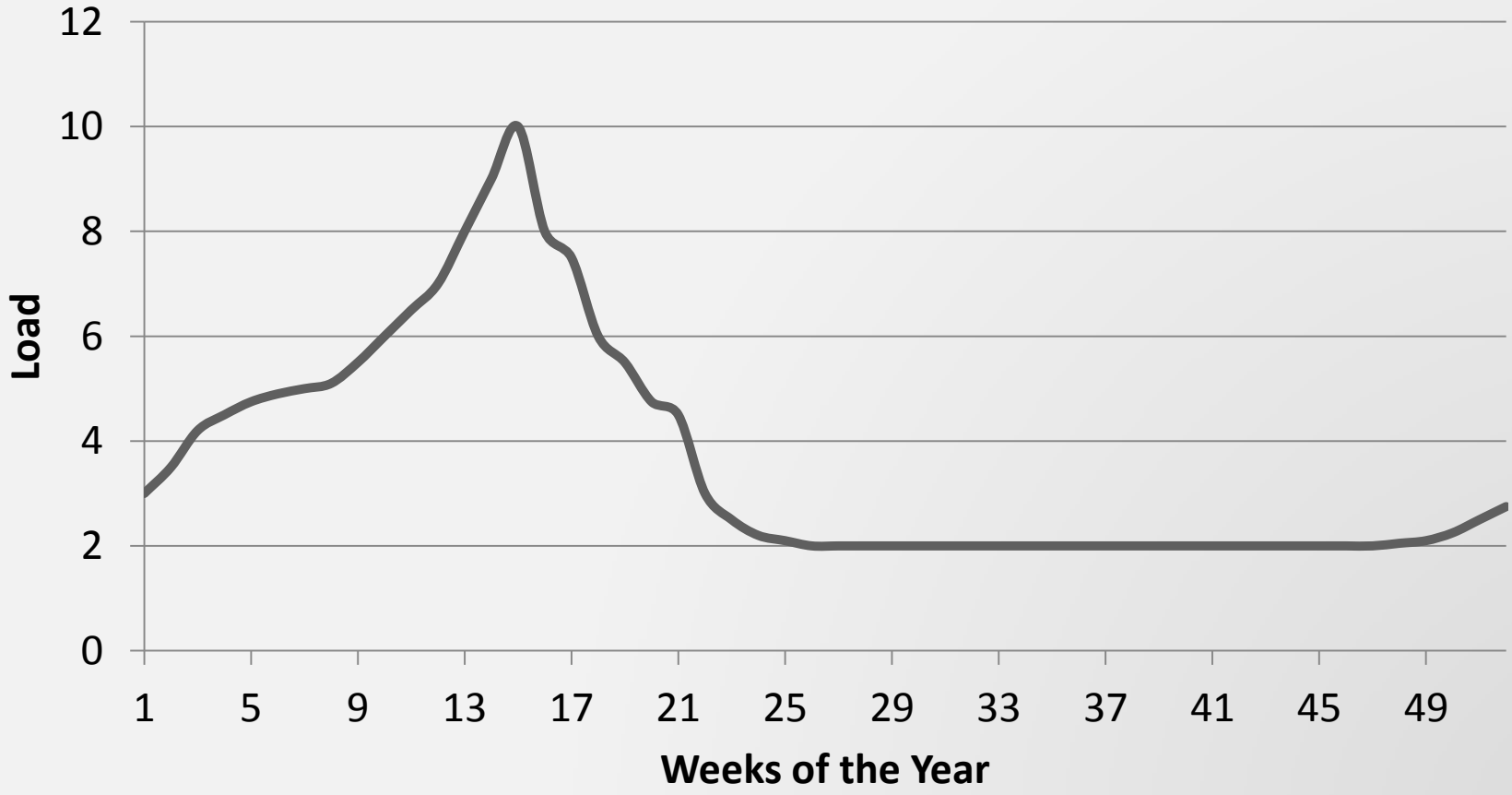
Expected Load Variation



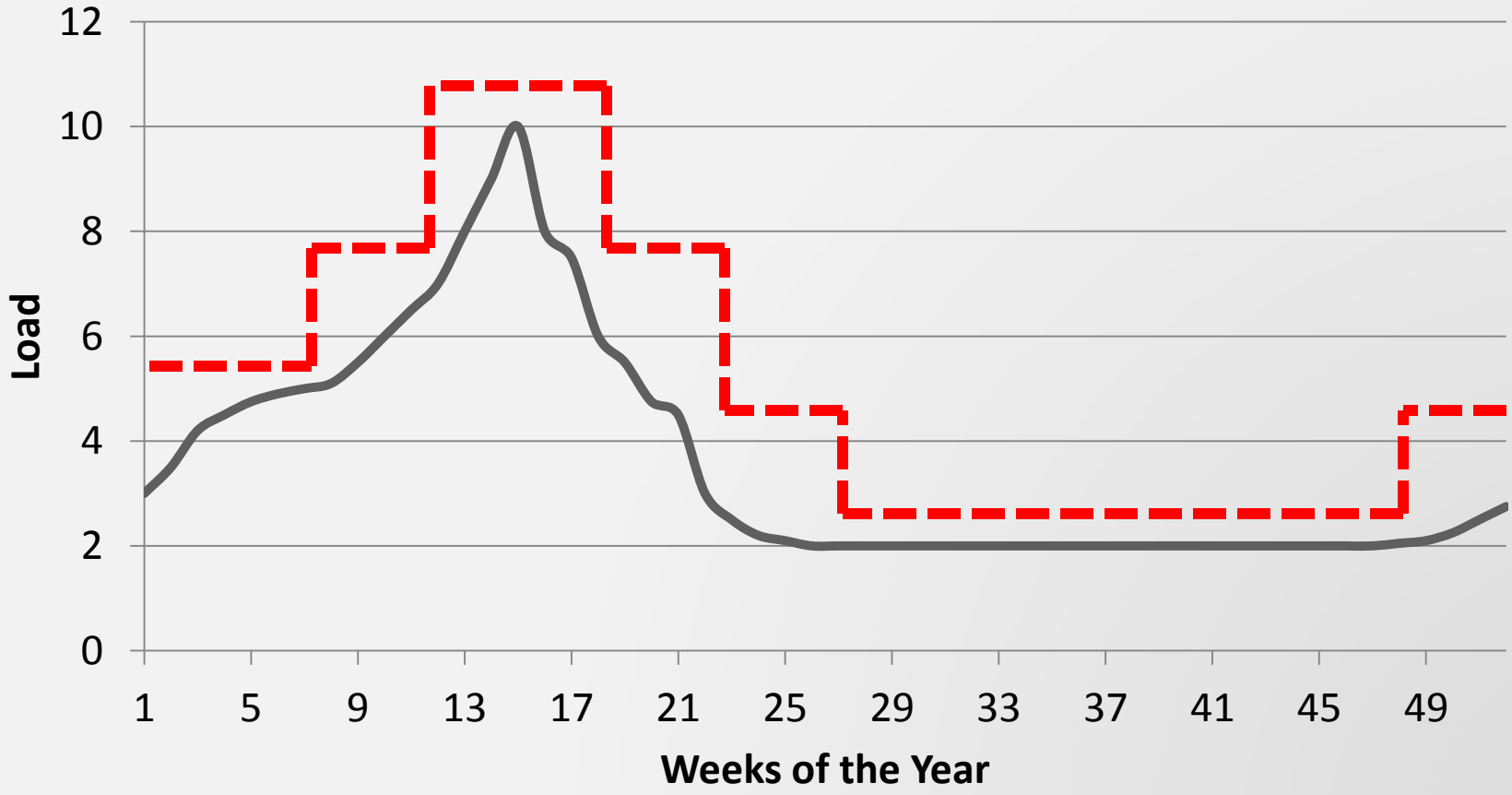
Elastic environments are:

- Highly utilized all the time
- Provision resources “just in time”

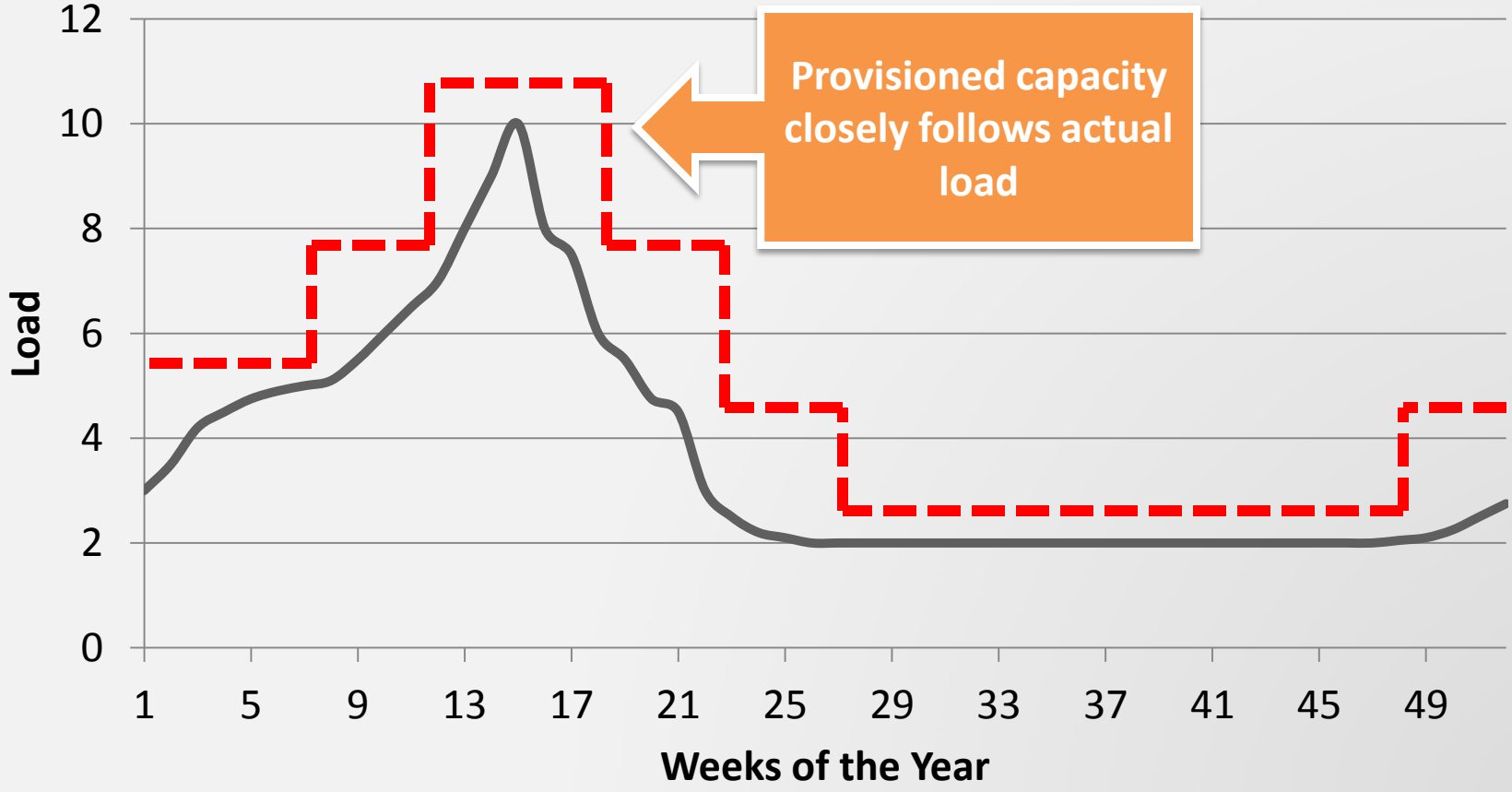
Expected Load Variation



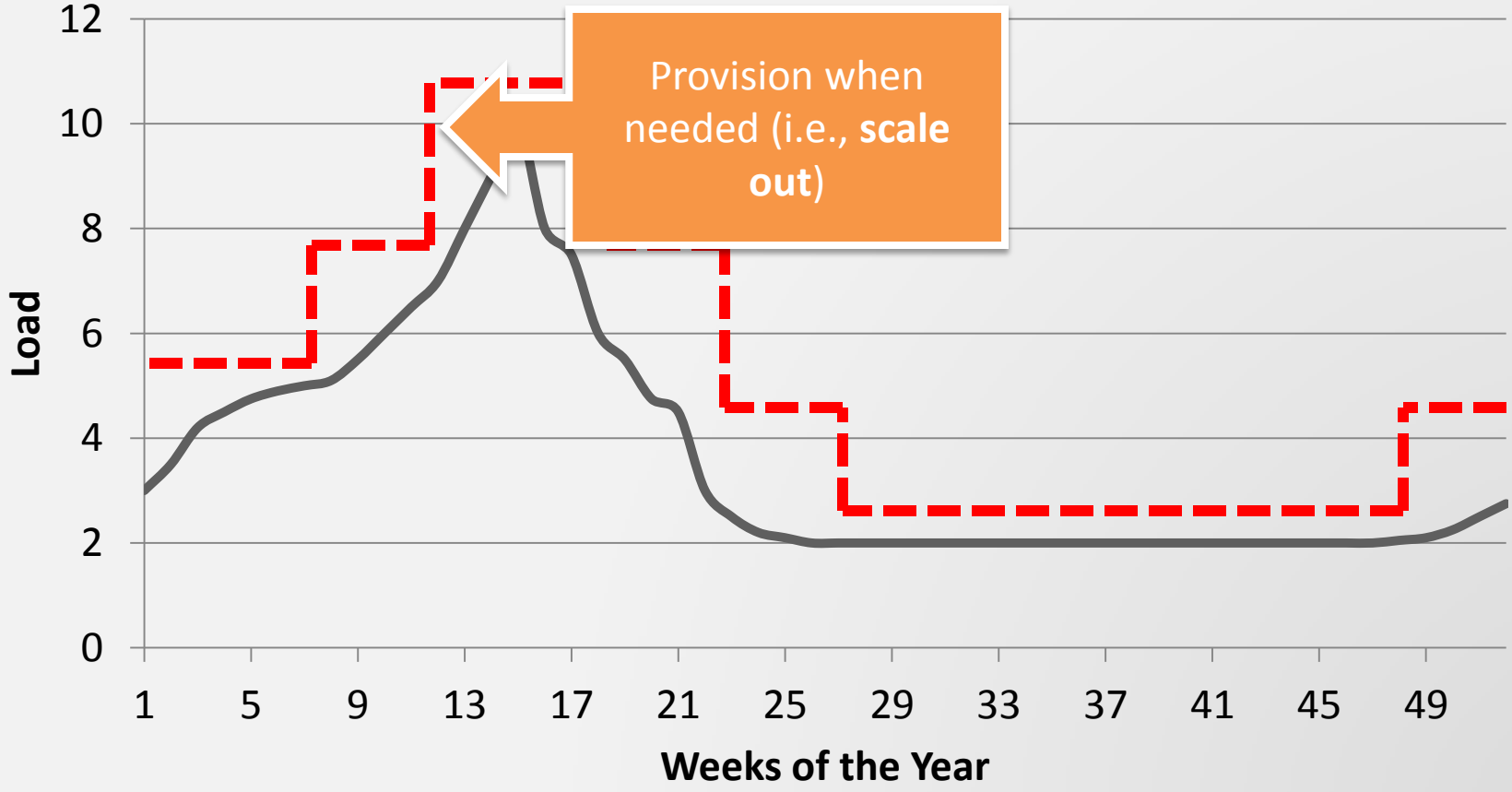
Expected Load Variation



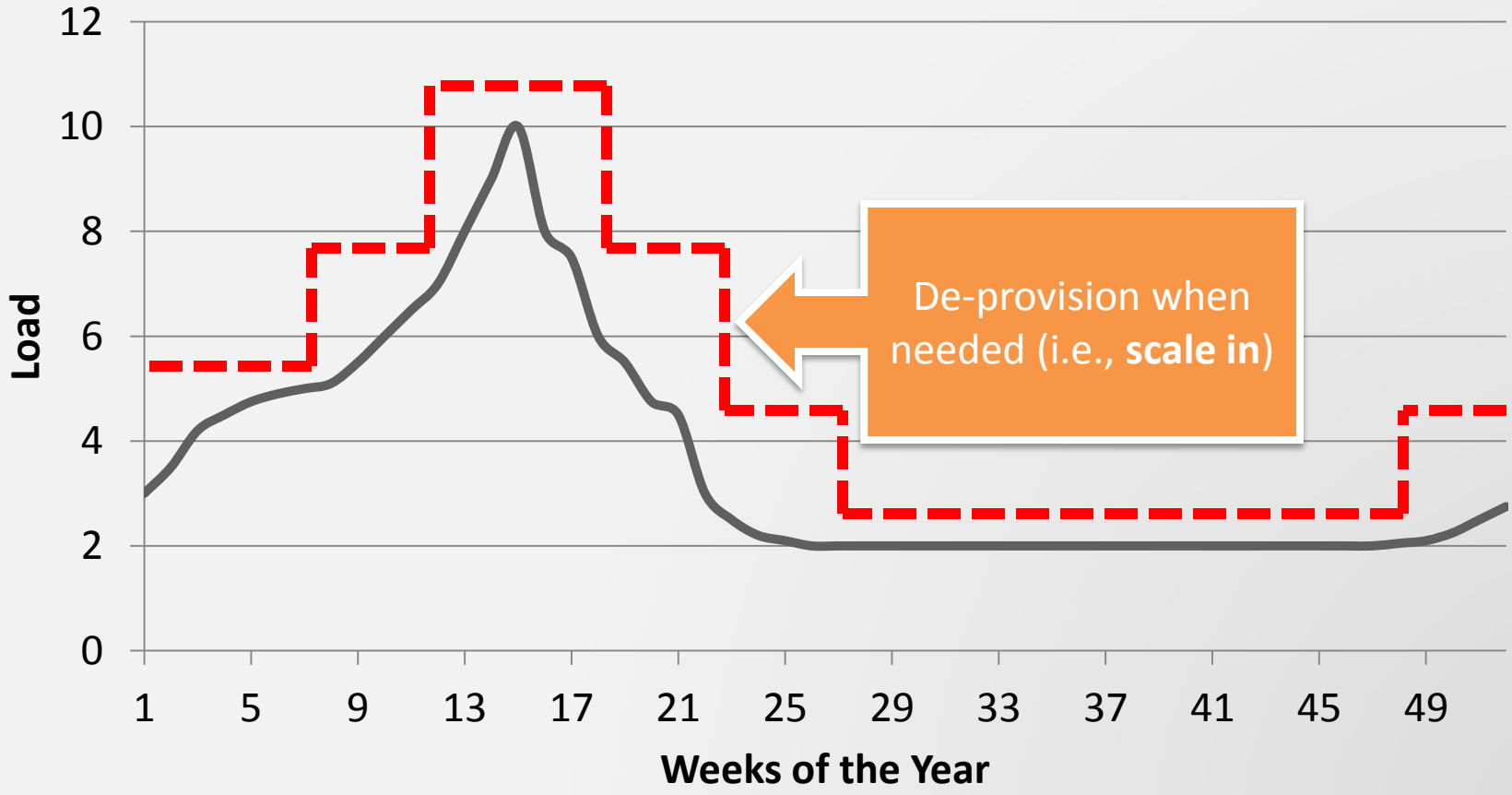
Expected Load Variation



Expected Load Variation



Expected Load Variation



2

Patterns and (anti-patterns) for creating scalable architectures in AWS

Elasticity and Scalability: Patterns and Anti-Patterns

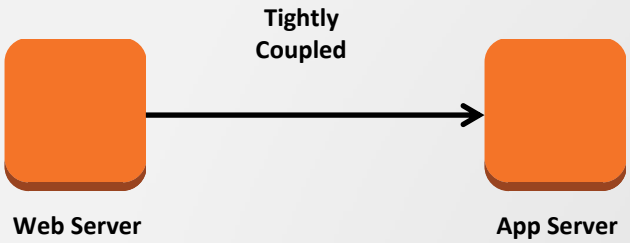
Anti-Pattern: **Manual Processes**

- When direct, manual intervention is required to start new resources—or scale existing ones—will be a blocker at scale

Pattern: **Automated Processes**

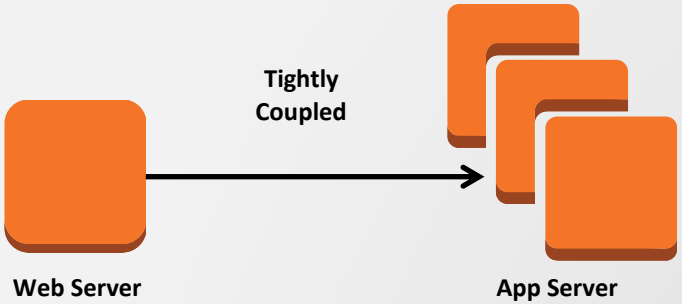
Anti-Pattern: **Tightly-coupled**

Application components in which a single unit depends on another *specific single unit* behave poorly when the dependency fails or needs to subdivide (for example, grow horizontally) to scale.

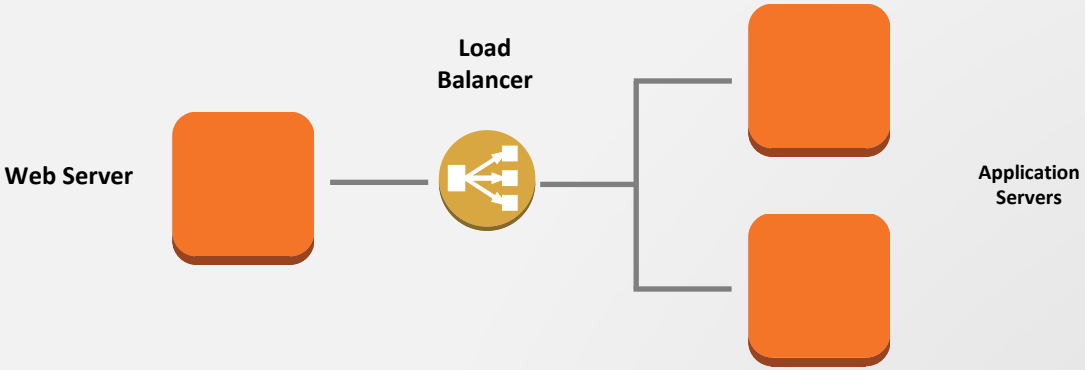


Anti-Pattern: **Tightly-coupled**

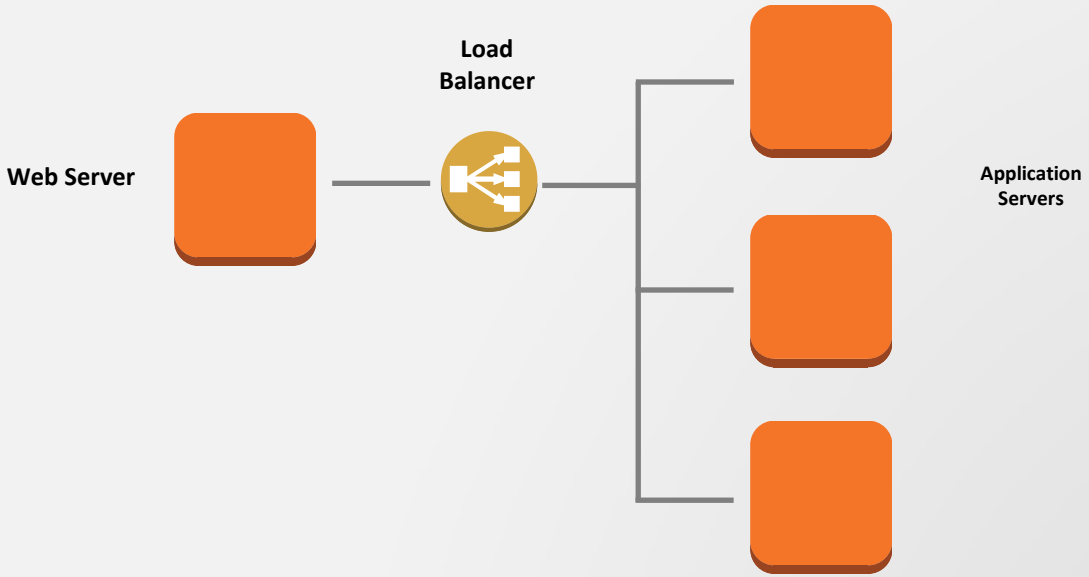
Application components in which a single unit depends on another *specific single unit* behave poorly when the dependency fails or needs to **subdivide (for example, grow horizontally)** to scale.



Pattern: **Loosely-coupled**

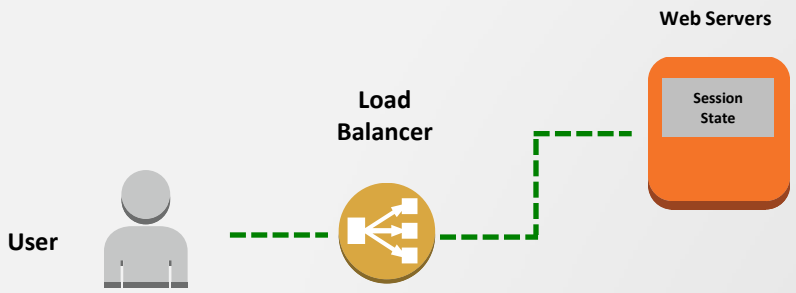


Pattern: **Loosely-coupled**



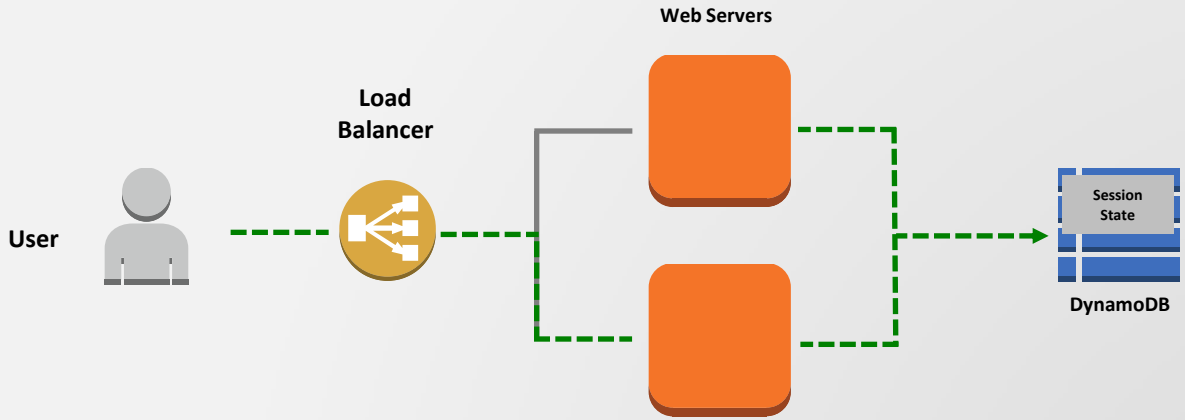
Anti-Pattern: **Stateful**

Applications that store state on one instance are more challenging to scale horizontally.



Pattern: **Stateless**

Move state to a shared, accessible location



Anti-Pattern: **Vertical**

Vertical scaling (more CPU, memory, etc) will eventually run out of room.

Anti-Pattern: **Vertical**

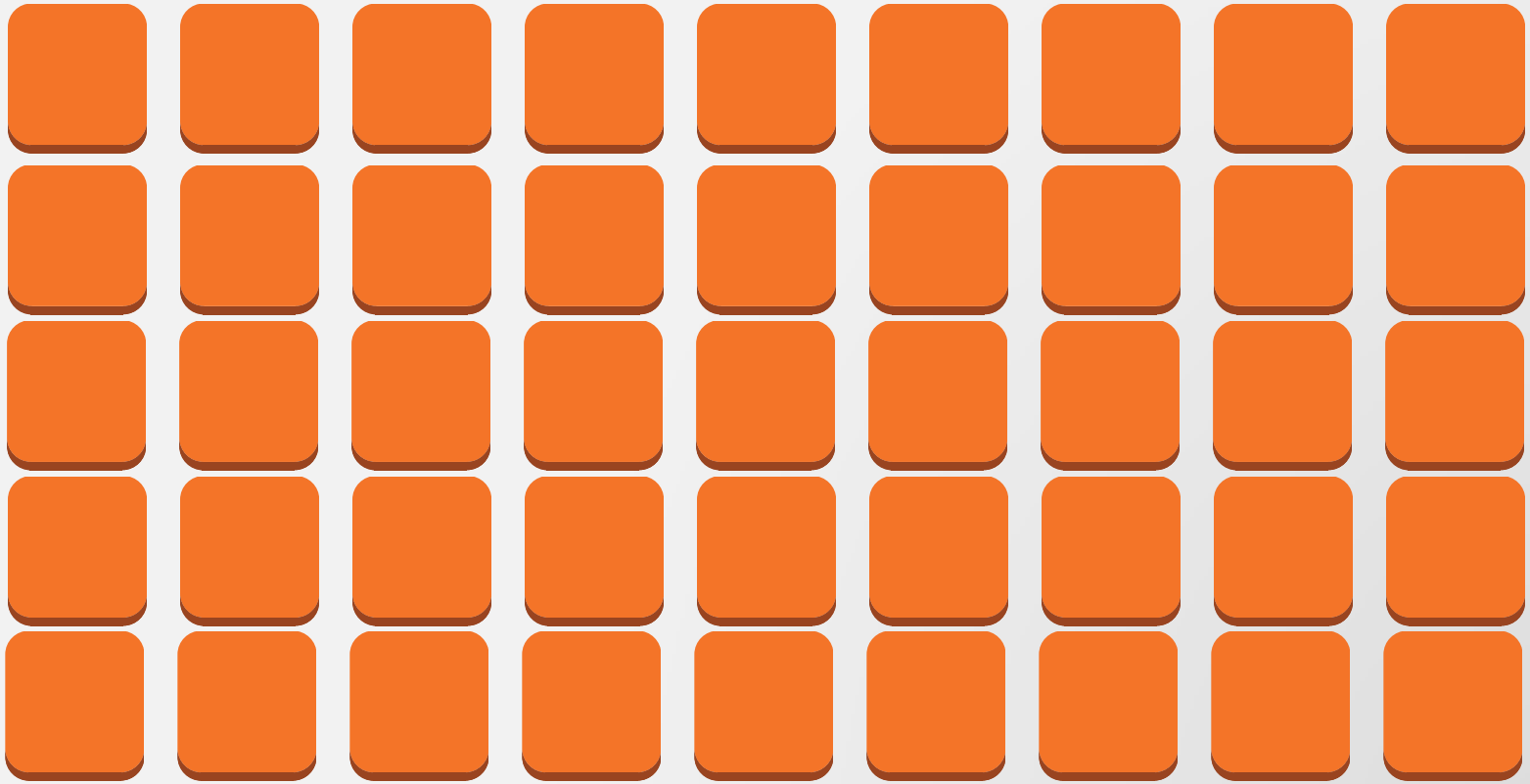
Vertical scaling (more CPU, memory, etc) will eventually run out of room.

Pattern: **Horizontal**

Add and remove instances as needed

Pattern: **Horizontal**

Add and remove instances as needed



3

Bootstrapping EC2 Instances

Bootstrapping

The process of **automatically setting up your servers**

Bootstrapping: some examples

- Install latest software
- Copy data from S3
- Register with DNS
- Start services
- Update packages
- Reboot
- Open port 80
- Register with load balancer
- Mount devices

Bootstrapping Tools

- **Scripts** on instance (Bash, Powershell)
- **Config Management Tools** (Chef, Puppet)

EC2 Metadata and UserData

- Every **EC2 Instance** has access to local instance **metadata** and **userdata service**

EC2 Metadata and UserData

- **Metadata**: immutable information about the instance

Accessible from within the instance via HTTP at

- **http://169.254.169.254/latest/meta-data/**

EC2 Metadata and UserData

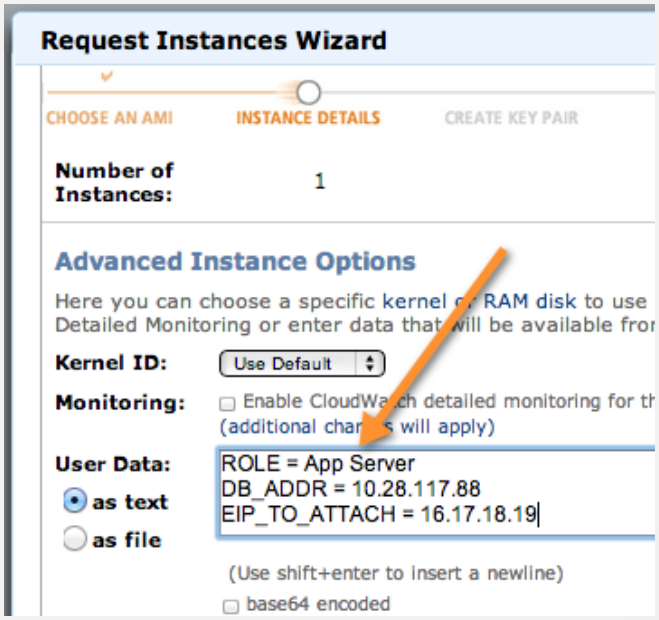
Script(s) on instance may retrieve useful information about the instance, such as:

- **Host name**
- **AMI ID**
- **Instance ID**
- **Public/Private DNS**
- **Availability Zone**

EC2 Metadata and **UserData**

- **Pass up to 16KB of text to an instance on launch**
- Text can be parsed by script on instance and used to configure the machine

EC2 Metadata and UserData



Custom script on AMI (e.g., script_runner.py) parses userdata and configures EC2 Instance on boot

UserData and CloudInit

- CloudInit executes UserData on first boot if UserData begins with:
  - **#! (Linux)**
  - **<script> (Windows; technically, EC2Config, not CloudInit, does this)**

UserData and CloudInit

- CloudInit executes UserData on first boot if UserData begins with:
  - **#! (Linux)**
  - **<script> (Windows; technically, EC2Config, not CloudInit, does this)**
- CloudInit is installed on Amazon Linux, Ubuntu, and RHEL AMIs
- EC2Config is installed on Windows Server AMIs
- Both may be installed on other distributions via a package repo or source

UserData and CloudInit

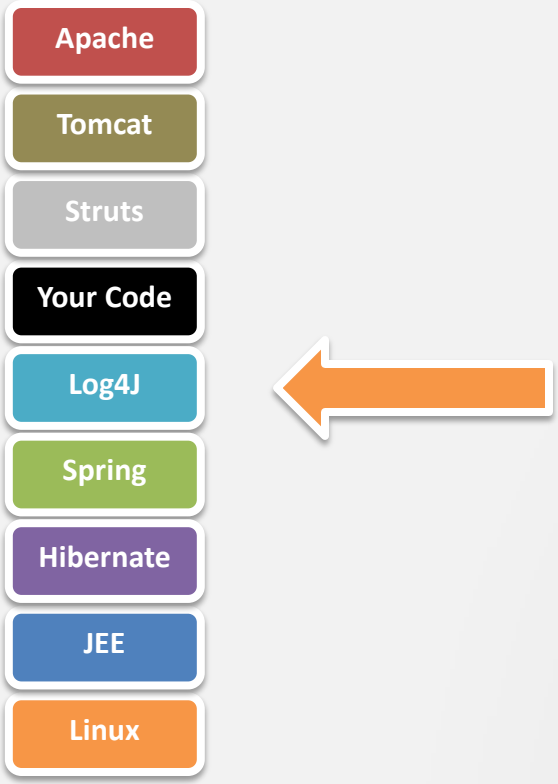
- UserData to **install Apache** and **MySQL on boot,** and **attach an EIP:**

```
#!/bin/bash

# Install Apache, PHP, and MySQL
yum install -y httpd mysql-server

# Attach an Elastic IP to this instance
ec2-associate-address \
  23.34.45.56 \
  -i $(curl http://169.254.169.254/latest/meta-data/instance-id)
```

3 Major Ways to Bootstrap AMIs

- Fully-Functional
- Partially Configured
- Base OS, Config with Code

AMIs and Bootstrapping

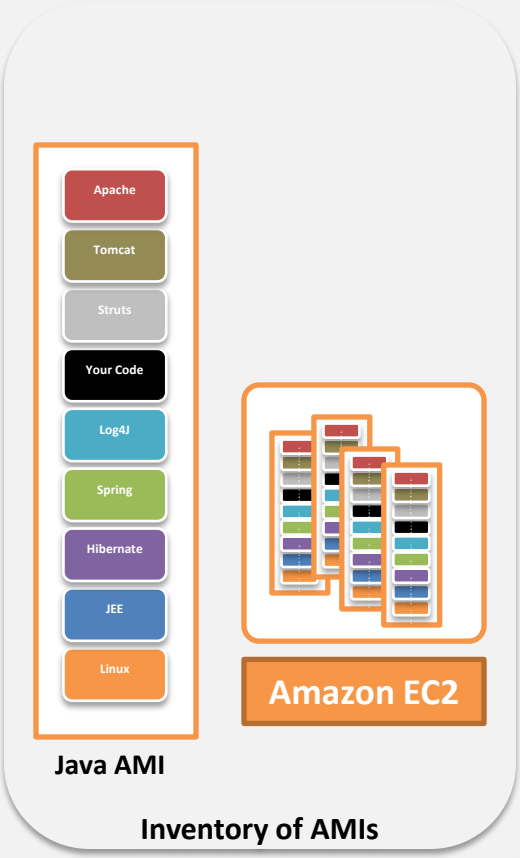


Java App Stack

Example full stack required to run your application.

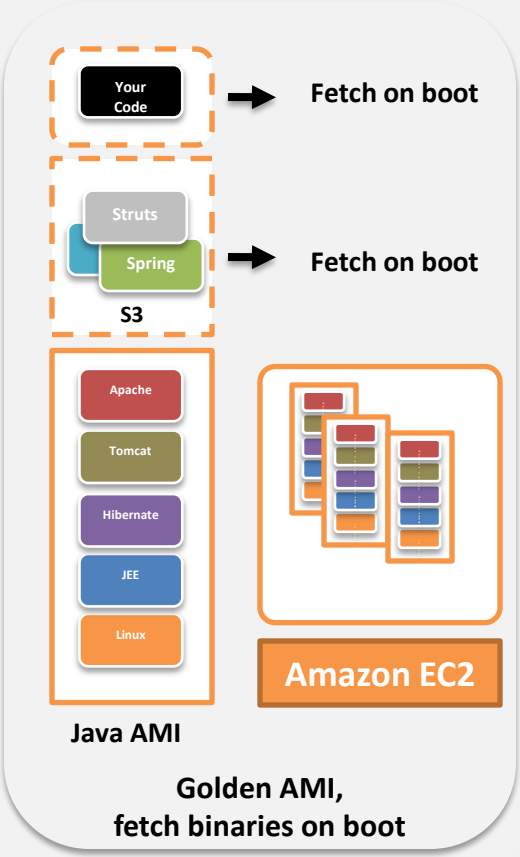
Let's use the 3 AMI/bootstrapping techniques to configure

Fully-functional AMI



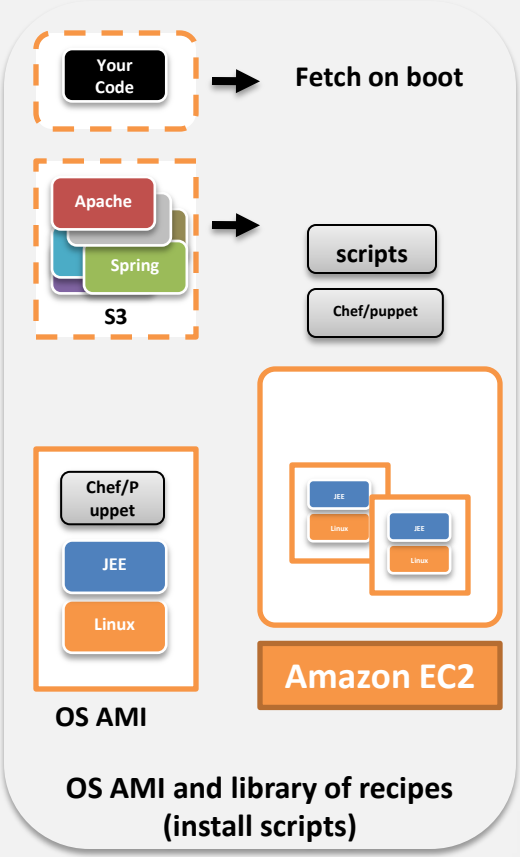
Fully-functional AMI is pre-build and ready to launch from the AMI inventory

Partially-configured AMI



A "Golden Image" is launched, with scripts fetching/installing app code and other supporting components on boot

Base OS AMI



An AMI with minimal components (e.g., OS, J2EE, and Chef/Puppet) is launched. All configuration occurs via Chef/Puppet after instance launch

4

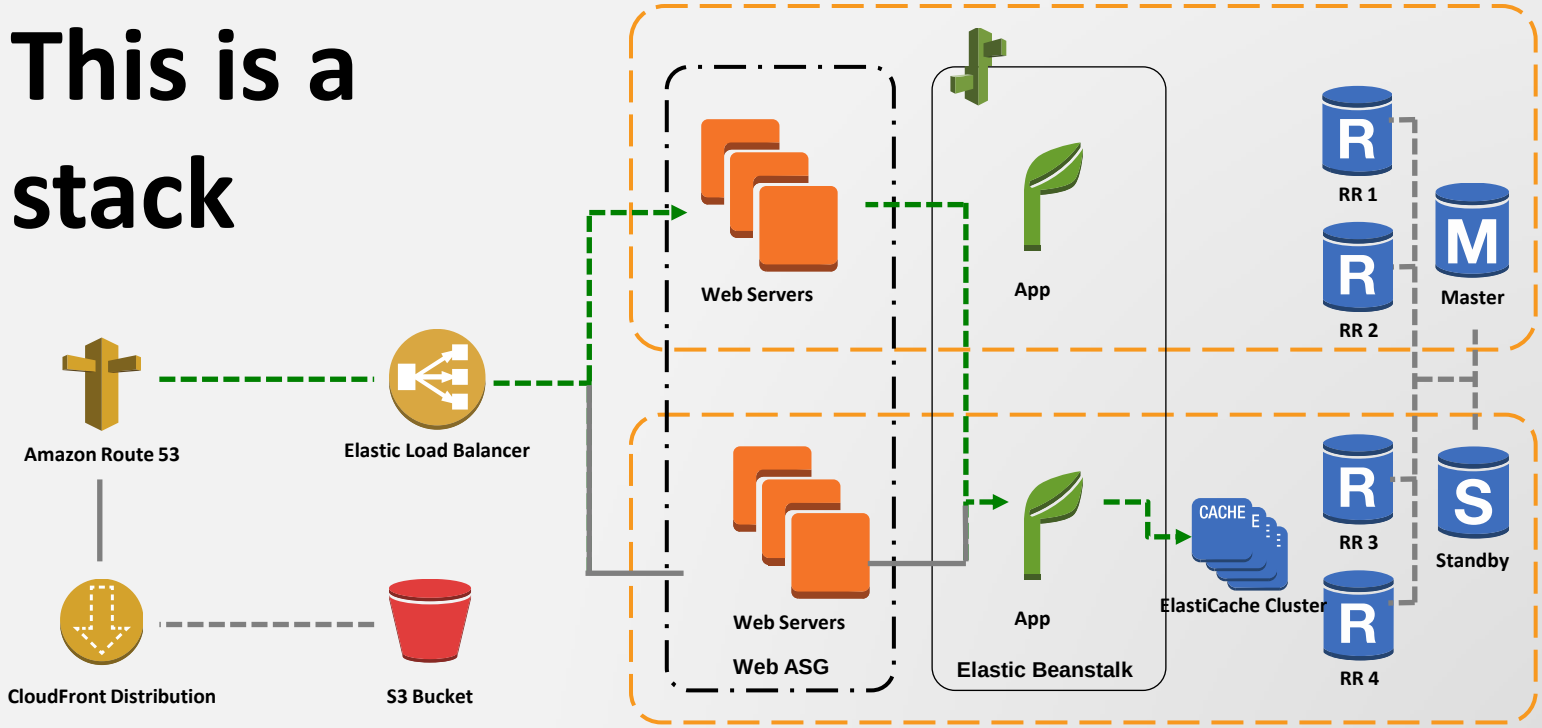
Building with CloudFormation

CloudFormation

- **Infrastructure as code**, suitable for change management in version control (e.g., git, svn, etc.)
- Define an entire **application stack** (i.e., all resources required for your application) in a JSON **template** file
- **Define runtime parameters** for a template (e.g., EC2 Instance Size, EC2 Key Pair, etc)
- **Generate templates** from running environments with CloudFormer

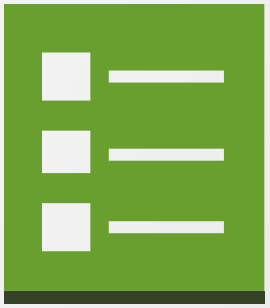
CloudFormation

This is a stack

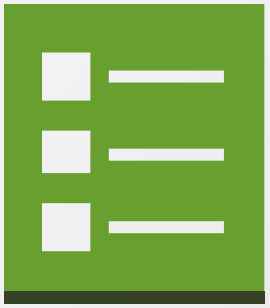


CloudFormation

This is a template file describing the stack



CloudFormation



Git
Subversion
Mercurial

CloudFormation

CloudFormation Template Anatomy

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : “my-key-pair”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : “my-key-pair”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

Resources are the AWS things you want to create

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : “my-key-pair”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

Logical resource name, anything you like

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.",
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : “my-key-pair”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

The type of the resource to create

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : “my-key-pair”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

Properties define how CloudFormation will call the ec2-run-instance API

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : “my-key-pair”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

We should allow the user of this template to specify her own EC2 Key Pair rather than hard-coding it.

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
       "UserKeyName" : {
          "Description" : "The EC2 Key Pair to allow SSH access to the instance",
          "Type" : "String"
       }
    },
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : { “Ref” : “UserKeyName”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

Parameters allow user of template to provide input

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
        "UserKeyName" : {
            "Description" : "The EC2 Key Pair to allow SSH access to the instance",
            "Type" : "String"
        }
    },
    "Resources" : {
        "Ec2Instance" : {
            "Type" : "AWS::EC2::Instance",
            "Properties" : {
                "KeyName" : { “Ref” : “UserKeyName”,
                "ImageId" : "ami-75g0061f”,
                “InstanceType” : “m1.medium”
            }
        }
    }
}
```

The KeyName property now references the UserKeyName parameter

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
       "UserKeyName" : {
          "Description" : "The EC2 Key Pair to allow SSH access to the instance",
          "Type" : "String"
       }
    },
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : { “Ref” : “UserKeyName”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : “m1.medium”
          }
       }
    }
}
```

Let’s also let the user choose Instance Type, but with some restrictions…

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
       "UserKeyName" : {
          "Description" : "The EC2 Key Pair to allow SSH access to the instance",
          "Type" : "String"
       },
       “InstanceType” : {
         “Description” : “The EC2 Instance Type to launch.”,
         “Type” : “String”,
         “AllowedValues” : [“t1.micro”, “m1.small”, “m1.medium”]
       }
    },
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : { “Ref” : “UserKeyName”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : { “Ref” : “InstanceType” }
          }
       }
    }
}
```

The InstanceType parameter defines the AllowedValues the user may provide when creating a stack from this template

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
       "UserKeyName" : {
          "Description" : "The EC2 Key Pair to allow SSH access to the instance",
          "Type" : "String"
       },
       “InstanceType” : {
         “Description” : “The EC2 Instance Type to launch.”,
         “Type” : “String”,
         “AllowedValues” : [“t1.micro”, “m1.small”, “m1.medium”]
       }
    },
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : { “Ref” : “UserKeyName”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : { “Ref” : “InstanceType” }
          }
       }
    }
}
```

Finally, we want to know the public DNS of the EC2 Instance that CloudFormation creates

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
       "UserKeyName" : {
          "Description" : "The EC2 Key Pair to allow SSH access to the instance",
          "Type" : "String"
       },
       “InstanceType” : {
         “Description” : “The EC2 Instance Type to launch.”,
         “Type” : “String”,
         “AllowedValues” : [“t1.micro”, “m1.small”, “m1.medium”]
       }
    },
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : { “Ref” : “UserKeyName”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : { “Ref” : “InstanceType” }
          }
       }
    },
    "Outputs" : {
       "InstancePublicDnsName" : {
          "Description" : "The public DNS name of the newly created EC2 instance",
          "Value" : { ”Fn::GetAtt" : [ "Ec2Instance”, “PublicDnsName” ] }
       }
```

Outputs allow us to output information about the resources created in the template

CloudFormation Template Anatomy

```
{
    "Description" : "Create an EC2 instance.”,
    "Parameters" : {
       "UserKeyName" : {
          "Description" : "The EC2 Key Pair to allow SSH access to the instance",
          "Type" : "String"
       },
       “InstanceType” : {
         “Description” : “The EC2 Instance Type to launch.”,
         “Type” : “String”,
         “AllowedValues” : [“t1.micro”, “m1.small”, “m1.medium”]
       }
    },
    "Resources" : {
       "Ec2Instance" : {
          "Type" : "AWS::EC2::Instance",
          "Properties" : {
             "KeyName" : { “Ref” : “UserKeyName”,
             "ImageId" : "ami-75g0061f”,
             “InstanceType” : { “Ref” : “InstanceType”
          }
       }
    },
    "Outputs" : {
       "InstancePublicDnsName" : {
          "Description" : "The public DNS name of the newly created EC2 instance",
          "Value" : { ”Fn::GetAtt" : [ "Ec2Instance”, “PublicDnsName” ] }
       }
```

Fn::GetAtt allows us to retrieve the ‘PublicDnsName’ attribute of the ‘Ec2Instance’ resource

More CloudFormation

```
"UserData": {
        "Fn::Base64": {
          "Fn::Join": [
            "",
            [
              "#!/bin/bash -ex\n",
              "yum -y install git-core\n",
              "yum -y install php-pear\n",
              "pear install Crypt_HMAC2-1.0.0\n",
              "pear install HTTP_Request-1.4.4\n",
              "pear install aws/sdk\n",
```

Bootstrap with User Data

More CloudFormation

```
"AppDatabase": {"Type": "AWS::CloudFormation::Stack",
"Metadata": { … },
"Properties": {
   "TemplateURL": {
      "Fn::Join": [
         "/",
         [
            { … },
            "RDS_MySQL_55.template"
         ]
      ]
   },
```

Embed and re-use templates

CloudFormation Metadata and cfn-init

- Declare metadata dynamically configure instances
- Works on Linux and Windows Server instances

CloudFormation Metadata and cfn-init

```
"Ec2Instance": {
  "Metadata": {
    "AWS::CloudFormation::Init": {
      "config": {
        "sources" : {
          "/usr/local/bin/s3cmd" : "https://github.com/s3tools/s3cmd”
        },
        "packages": {
          "yum": { "git": [] }
        }
    }
  }
}
```

Define Instance Config

5

Components of Auto Scaling

Auto Scaling

- Scale your Amazon EC2 capacity up or down automatically according to conditions you define
- Ensure that the number of Amazon EC2 instances you're using increases seamlessly during demand spikes to maintain performance, and decreases automatically during demand lulls to minimize costs

Components

1. Launch configuration
2. Group
3. Scaling policy (optional)
4. Scheduled action (optional)

Auto Scaling Launch Configuration

- Define how Auto Scaling should launch your EC2 instances
- Similar to ec2-run-instances API

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

Created here using CLI

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

Launch Config name

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

AMI to launch

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

Instance Type to launch

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

EC2 Key Pair to use (optional)

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

Instance Security Group (optional)

Auto Scaling Launch Configuration

```
as-create-launch-config web-amzn-linux-m1.medium \
    --image-id ami-e565ba8c \
    --instance-type m1.medium \
    --key my-ec2-keypair \
    --group web-sg \
    --user-data bootstrap.sh
```

Instance User Data (optional)

Auto Scaling Group

- Define *where* (i.e., AZs or VPC Subnets) Auto Scaling should launch instances
- Optionally, ELB(s) to register with, tags to apply, etc.

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
      --availability-zones 'us-east-1a,us-east-1b'
      --launch-configuration 'web-amzn-linux-m1.medium'
      --min-size 2
      --max-size 8
      --desired-capacity 4
      --load-balancers 'elb-1'
```

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
      --availability-zones 'us-east-1a,us-east-1b'
      --launch-configuration 'web-amzn-linux-m1.medium'
      --min-size 2
      --max-size 8
      --desired-capacity 4
      --load-balancers 'elb-1'
```

Group name

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
    --availability-zones 'us-east-1a,us-east-1b'
    --launch-configuration 'web-amzn-linux-m1.medium'
    --min-size 2
    --max-size 8
    --desired-capacity 4
    --load-balancers 'elb-1'
```

Availability Zones to span

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
    --availability-zones 'us-east-1a,us-east-1b'
    --launch-configuration 'web-amzn-linux-m1.medium'
    --min-size 2
    --max-size 8
    --desired-capacity 4
    --load-balancers 'elb-1'
```

Launch Configuration to use

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
    --availability-zones 'us-east-1a,us-east-1b'
    --launch-configuration 'web-amzn-linux-m1.medium'
    --min-size 2
    --max-size 8
    --desired-capacity 4
    --load-balancers 'elb-1'
```

Minimum number of instances in Group

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
    --availability-zones 'us-east-1a,us-east-1b'
    --launch-configuration 'web-amzn-linux-m1.medium'
    --min-size 2
    --max-size 8
    --desired-capacity 4
    --load-balancers 'elb-1'
```

Maximum number of instances in Group

Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
    --availability-zones 'us-east-1a,us-east-1b'
    --launch-configuration 'web-amzn-linux-m1.medium'
    --min-size 2
    --max-size 8
    --desired-capacity 4
    --load-balancers 'elb-1'
```

Desired number of instances in Group (optional)

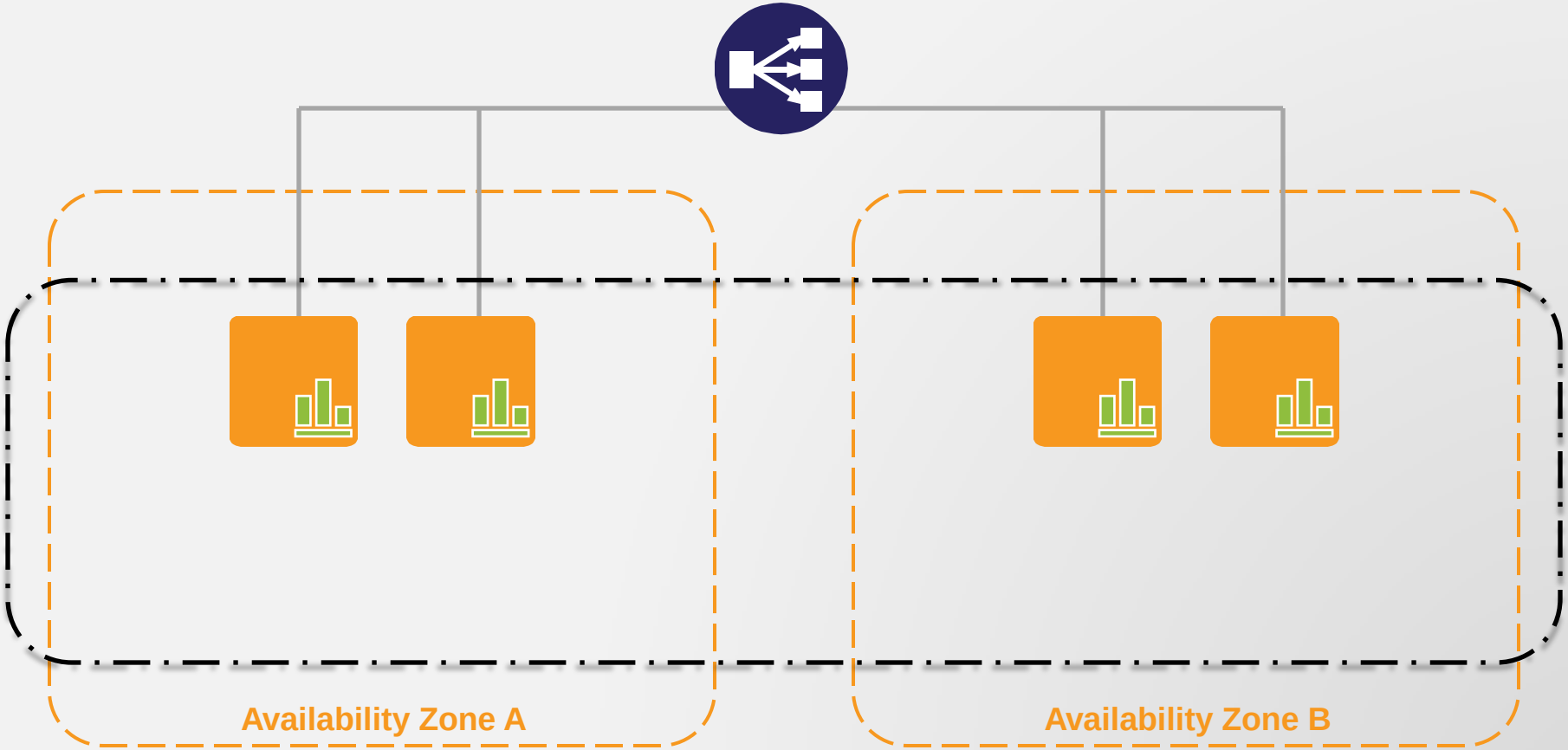
Auto Scaling Group

```
as-create-auto-scaling-group web-scaling-group
    --availability-zones 'us-east-1a,us-east-1b'
    --launch-configuration 'web-amzn-linux-m1.medium'
    --min-size 2
    --max-size 8
    --desired-capacity 4
    --load-balancers 'elb-1'
```

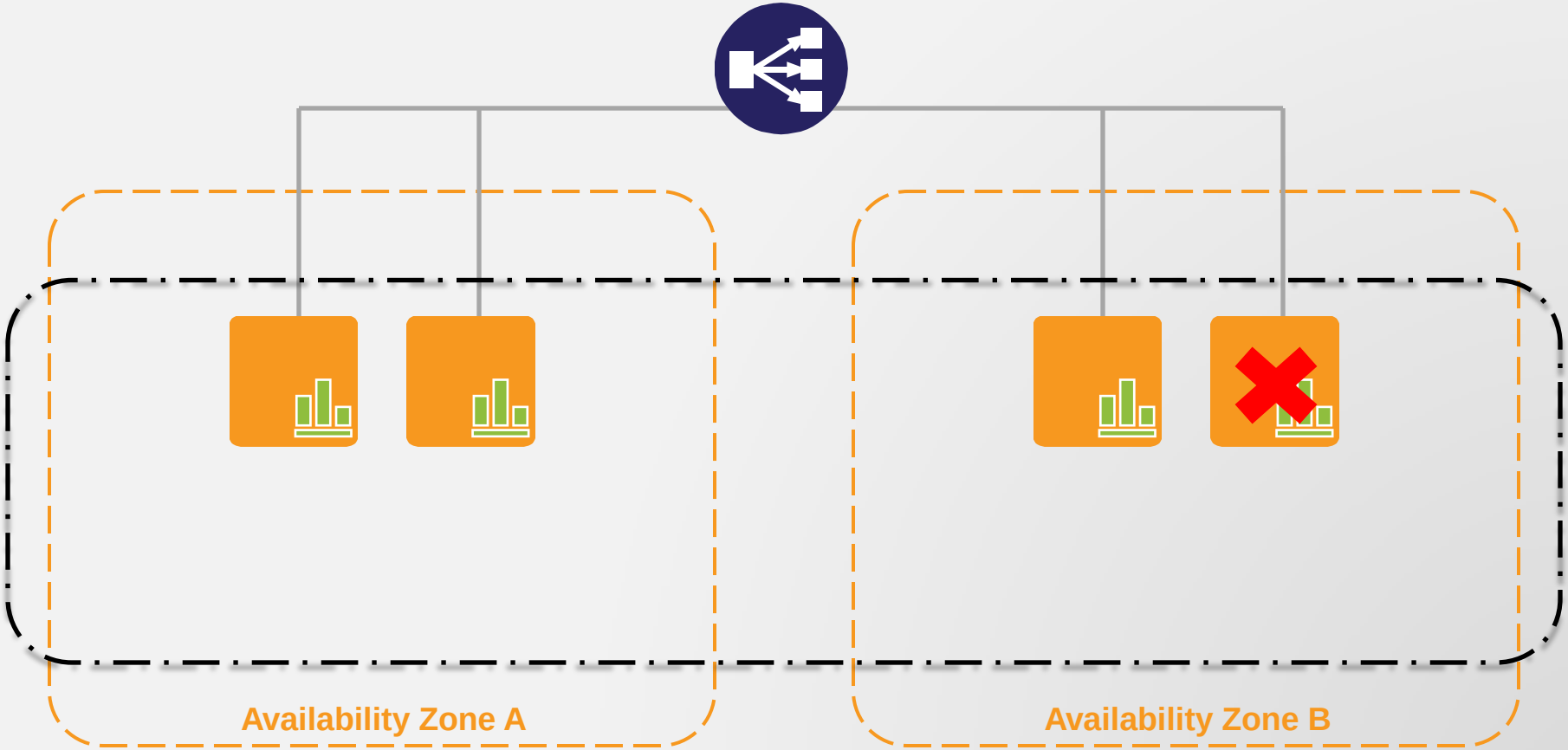
ELB(s) to register instances with (optional)

Auto Scaling Group

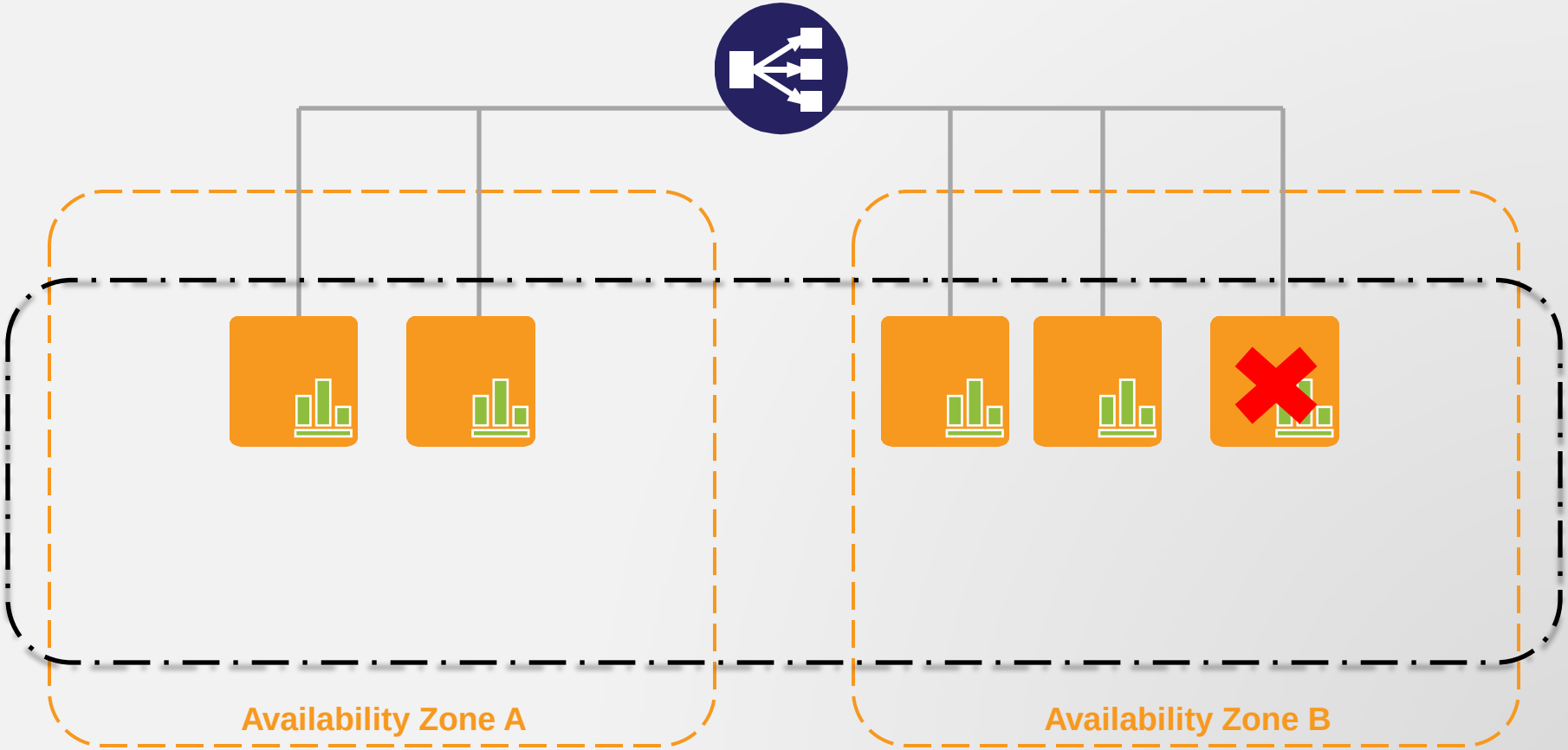
Result



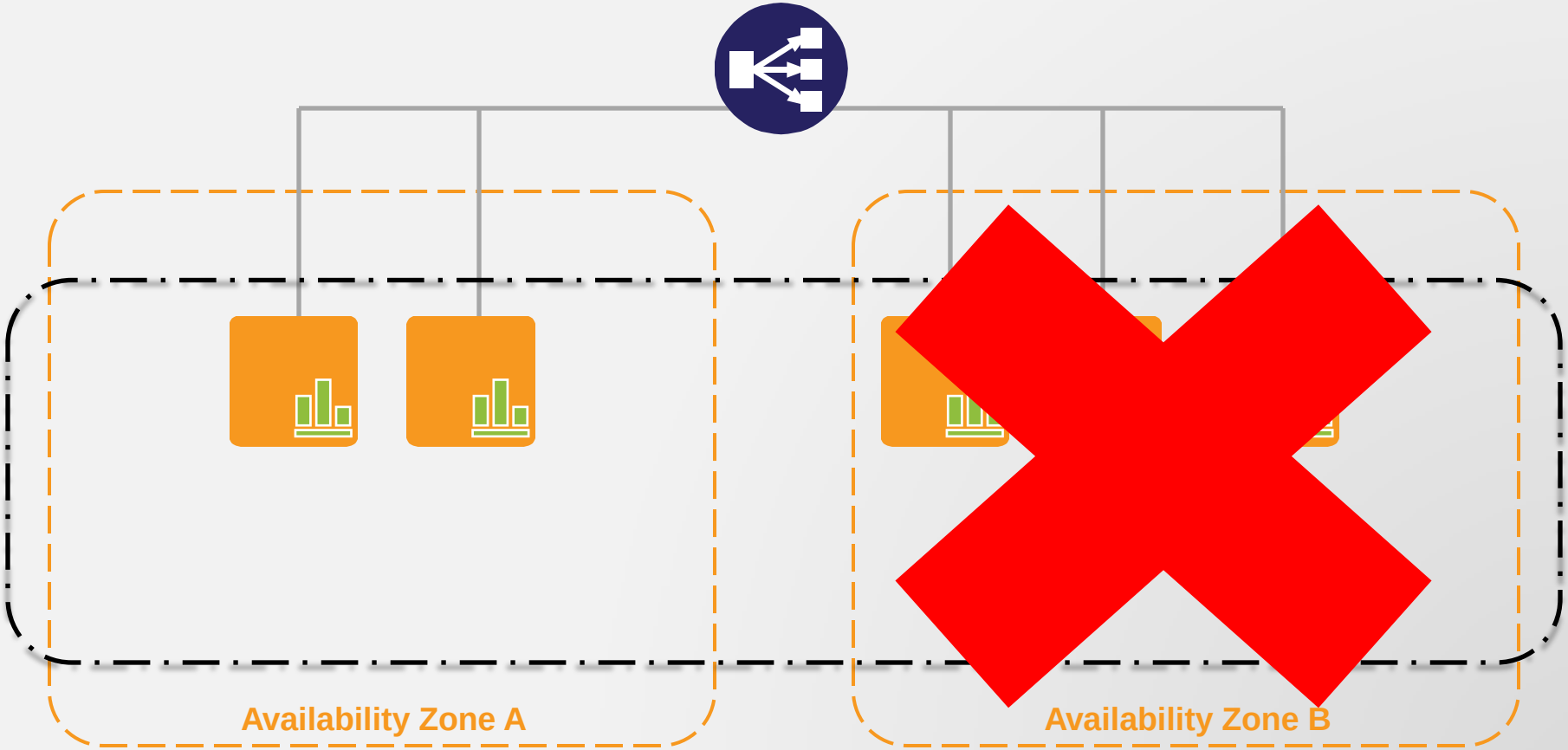
Auto Scaling Group



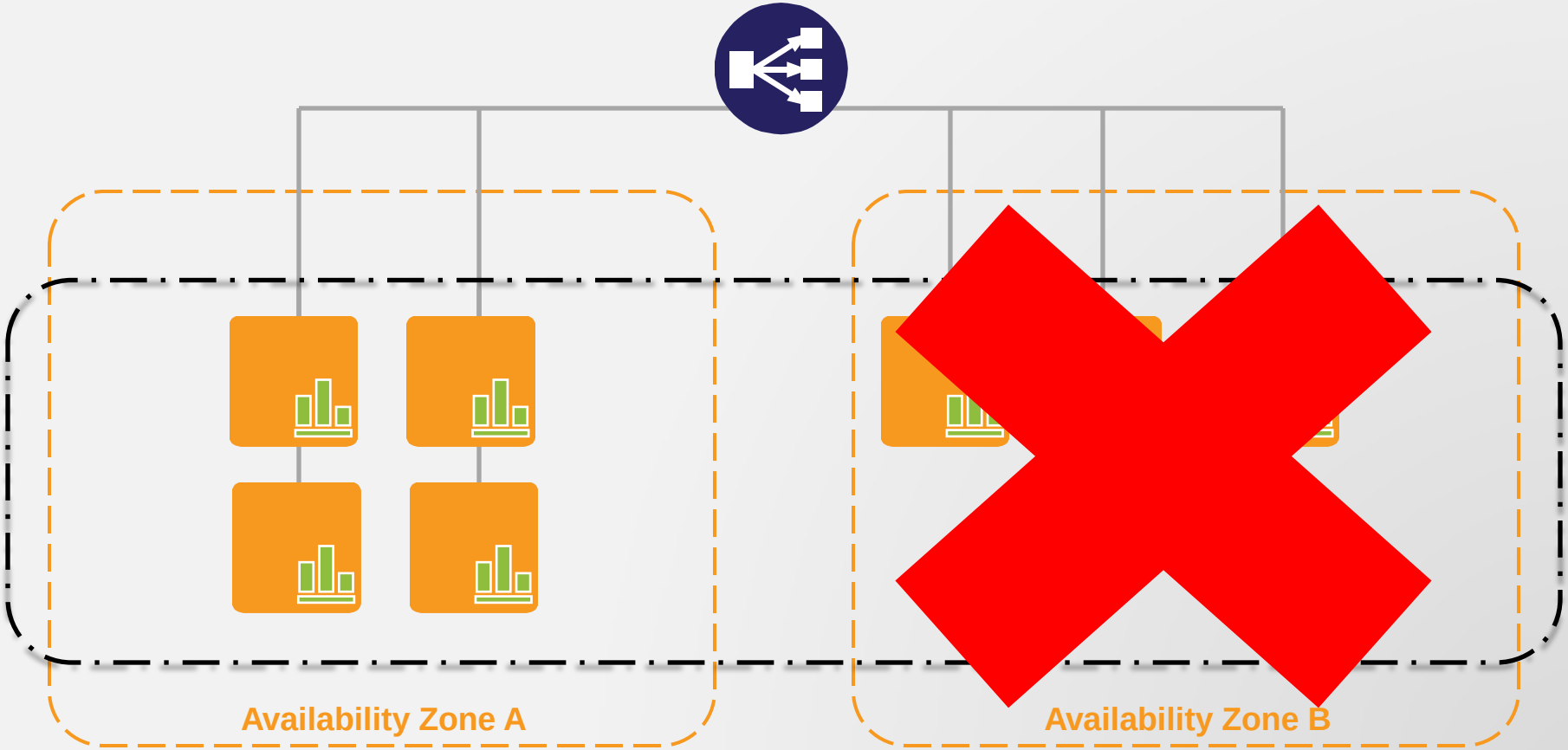
Auto Scaling Group



Auto Scaling Group



Auto Scaling Group



Auto Scaling Policy

- Define scaling actions that describes how many EC2 instances to add or remove to a Group

Auto Scaling Policy

```
as-put-scaling-policy web-remove-two-servers
    --auto-scaling-group web-scaling-group
    --adjustment '-2'
    --type ChangeInCapacity
```

Auto Scaling Policy

```
as-put-scaling-policy web-remove-two-servers
    --auto-scaling-group web-scaling-group
    --adjustment '-2'
    --type ChangeInCapacity
```

Policy name

Auto Scaling Policy

```
as-put-scaling-policy web-remove-two-servers
    --auto-scaling-group web-scaling-group
    --adjustment '-2'
    --type ChangeInCapacity
```

Group to affect

Auto Scaling Policy

```
as-put-scaling-policy web-remove-two-servers
    --auto-scaling-group web-scaling-group
    --adjustment '-2'
    --type ChangeInCapacity
```

Number of instances to add or remove

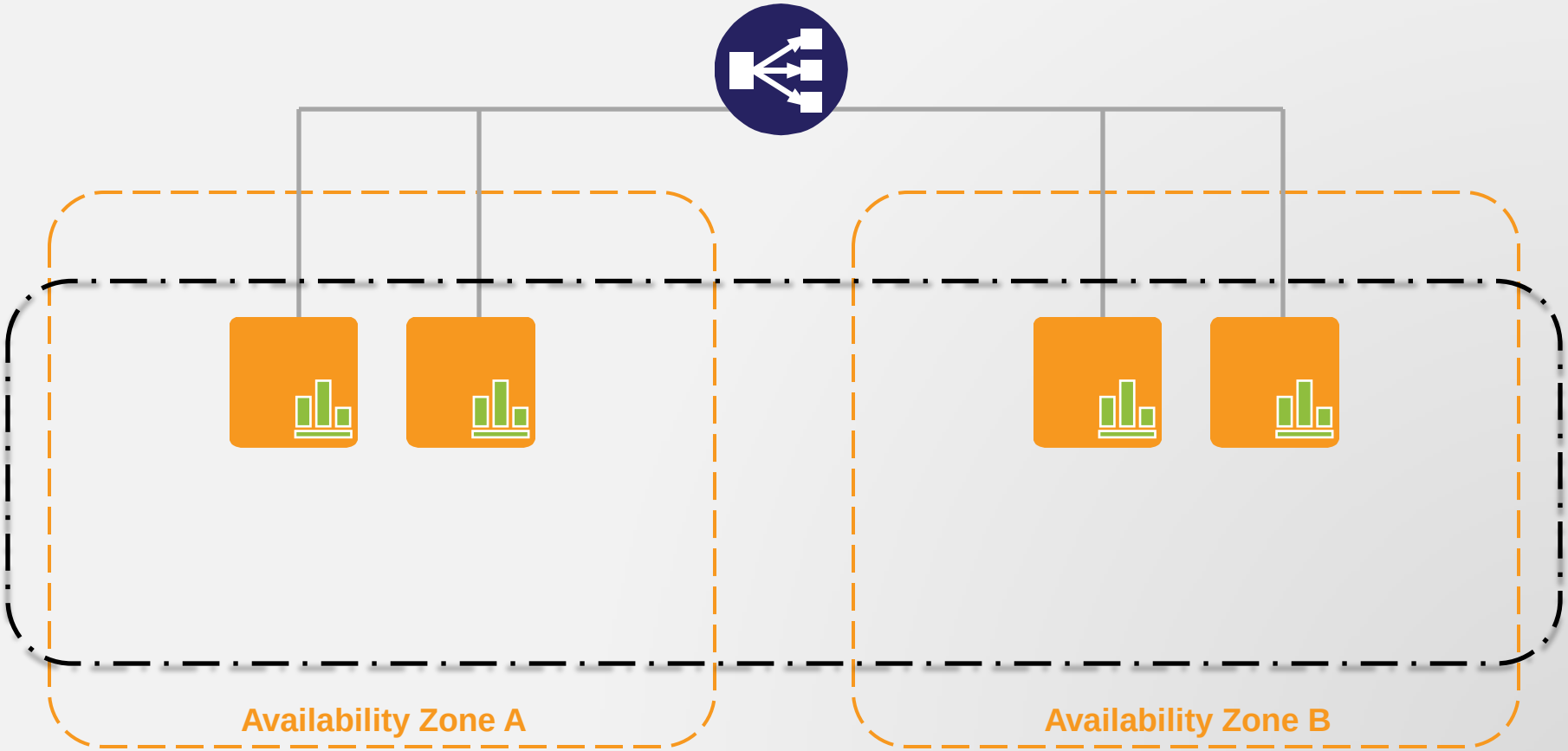
Auto Scaling Policy

```
as-put-scaling-policy web-remove-two-servers
    --auto-scaling-group web-scaling-group
    --adjustment '-2'
    --type ChangeInCapacity
```

Increment or decrement type

Auto Scaling Policy

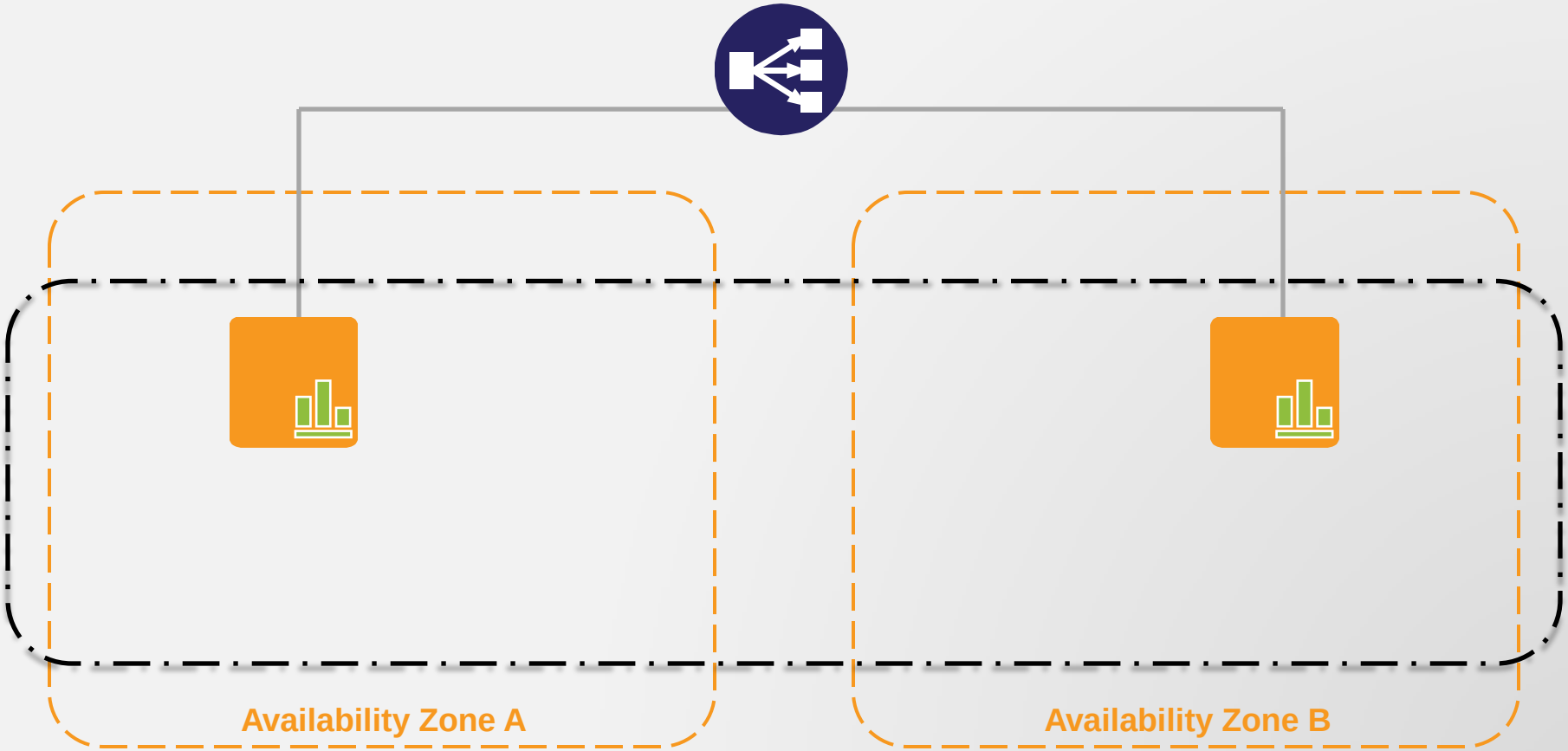
as-execute-policy web-remove-two-servers



Auto Scaling Policy

Result

as-execute-policy web-remove-two-servers



Auto Scaling Policy

```
as-put-scaling-policy web-double-servers
    --auto-scaling-group web-scaling-group
    --adjustment '100'
    --type PercentChangeInCapacity
```

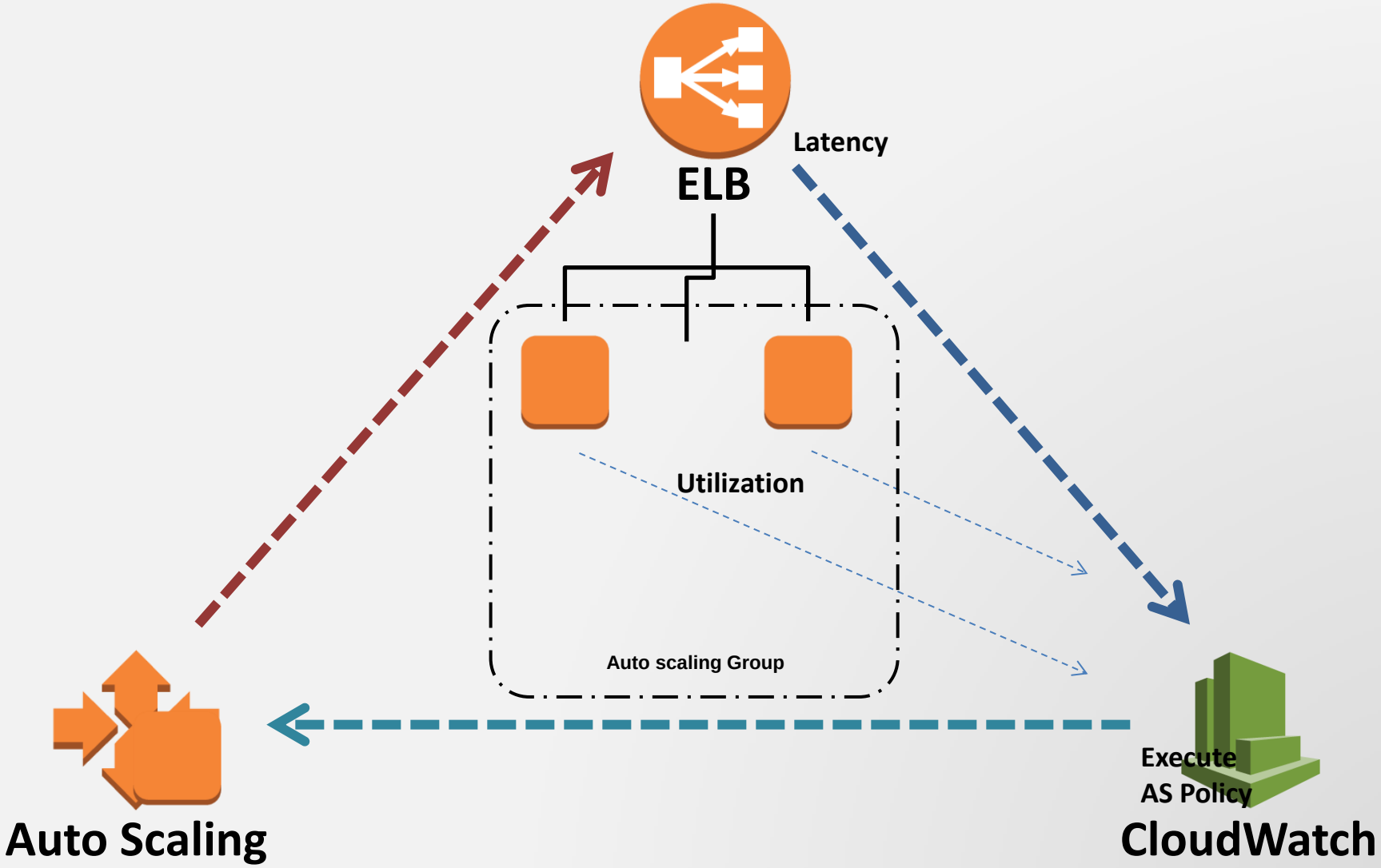
Percent Change in Capacity

Auto Scaling Policy

```
as-put-scaling-policy web-min-servers
    --auto-scaling-group web-scaling-group
    --adjustment '2'
    --type ExactCapacity
```

Fixed Capacity

ELB, CloudWatch, and Auto Scaling



For review

- What are the four patterns for building scalable architectures?
- Can you give five examples of bootstrapping an instance?
- How do you get the metadata for any given instance?
- How do you build a basic CloudFormation template?
- What are the four components of auto scaling?

Architecting with AWS

Elasticity, Scalability, and Bootstrapping

Appendix: Auto Scaling

Auto Scaling

- Automatically Scale Server Farms
  - Scale up ***and*** down
  - (Re)Balance Across AZs
  - Add/Remove from ELB if applicable

- Set a Thermostat
  - Don’t manage the furnace burners

Types of scaling

- Manual
  - Send an API call or use CLI to launch/terminate instances
  - Only need to specify capacity change (+/-)
- By Schedule
  - Scale up/down based on date and time
- By Policy
  - Scale in response to changing conditions, based on user configured real-time monitoring and alerts
- Automatic Rebalance
  - Instances are automatically launched/terminated to ensure the application is balanced across multiple AZs

Auto Scaling Components

- Launch Configuration
- Auto Scaling Group
- Auto Scaling Policy
- CloudWatch Alarms

Launch Configuration

- Describes what Auto Scaling will create when adding instances
  - Name (myLC)
  - AMI (ami-0535d66c)
  - Instance Type (m1.medium)
  - Security Group (SSH, Web, aws-elb-sg)
  - Instance Key Pair (myKeyPair)
- Only one active launch configuration at a time
- Auto Scaling will terminate instances with old launch configurations first
  - Rolling software updates

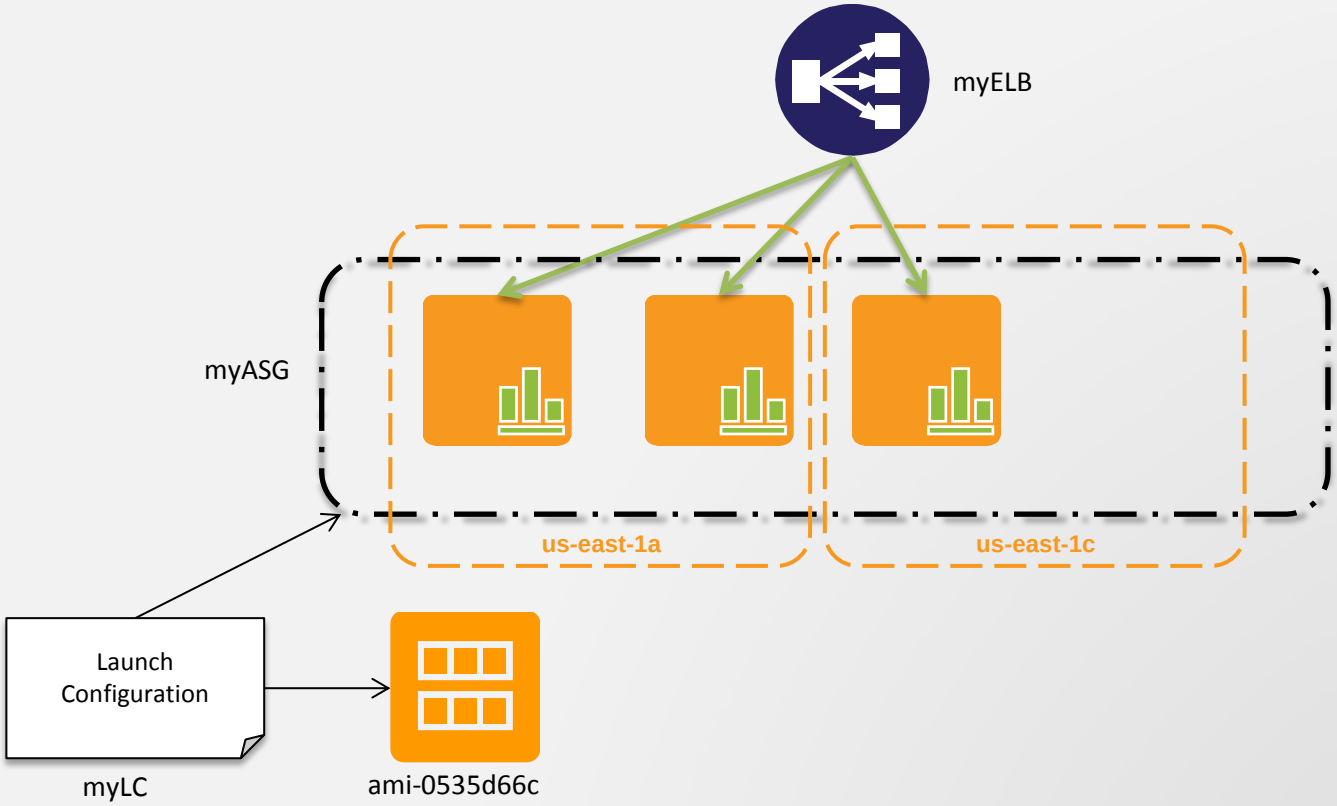
```
as-create-launch-config myLC --image-id ami-0535d66c --instance-type m1.medium –group SSH, Web, aws-elb-sg – key myKeyPair
```

Auto Scaling Group

- Auto Scaling managed grouping of EC2 instances
- Automatic health check to maintain pool size
- Automatically scale the number of instances by policy
  - Min, Max, Desired (how many initially)
- Automatic Integration with ELB
- Automatic Integration with AZs
  - Automatic distribution & balancing across AZs

```
as-create-auto-scaling-group myASG –launch-configuration myLC –availability-zones us-east-1a, us-east-1c
–min-size 1 –max-size 10 –desired-capacity 3 –load-balancers myELB
```

Auto Scaling

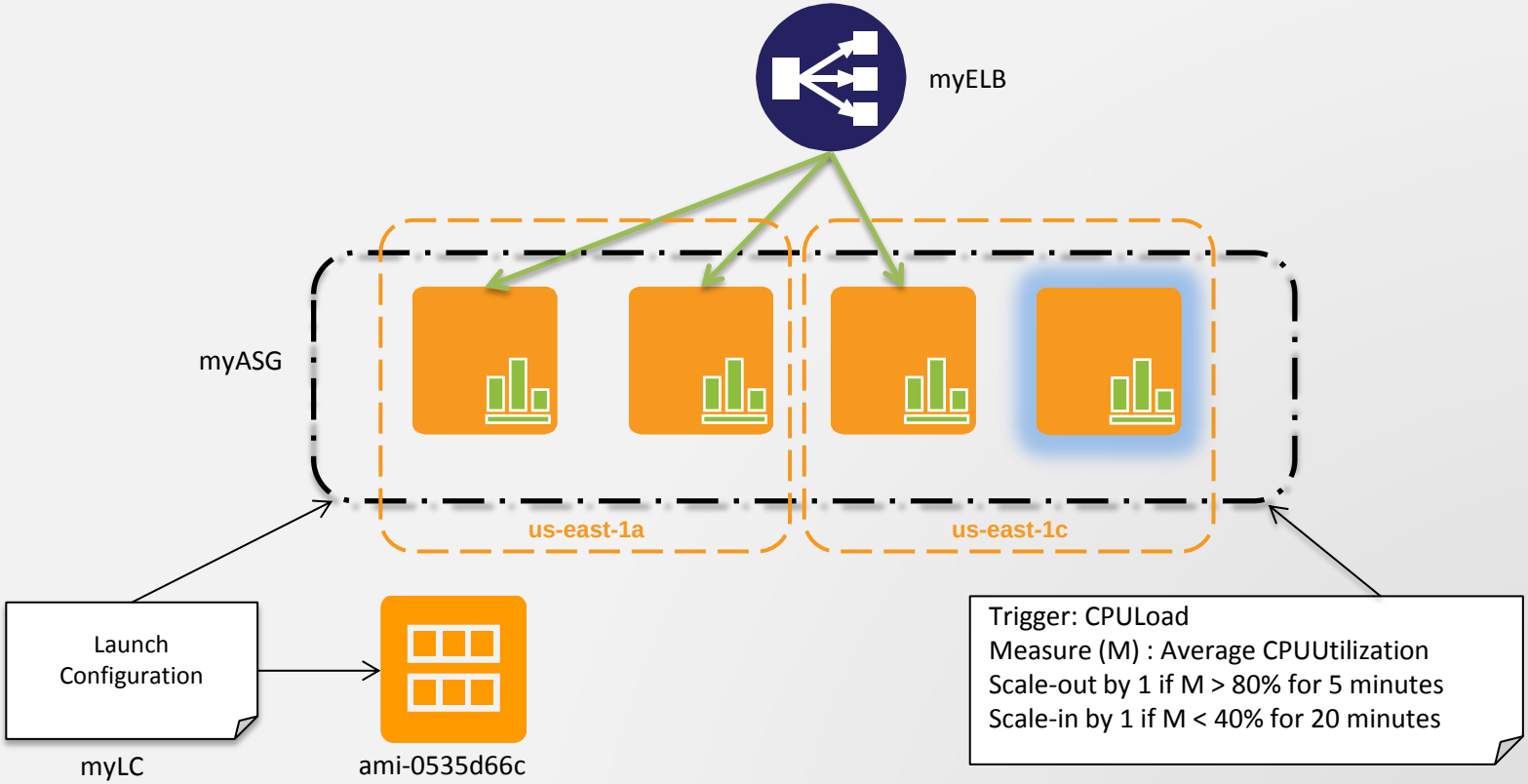


Auto Scaling Policy

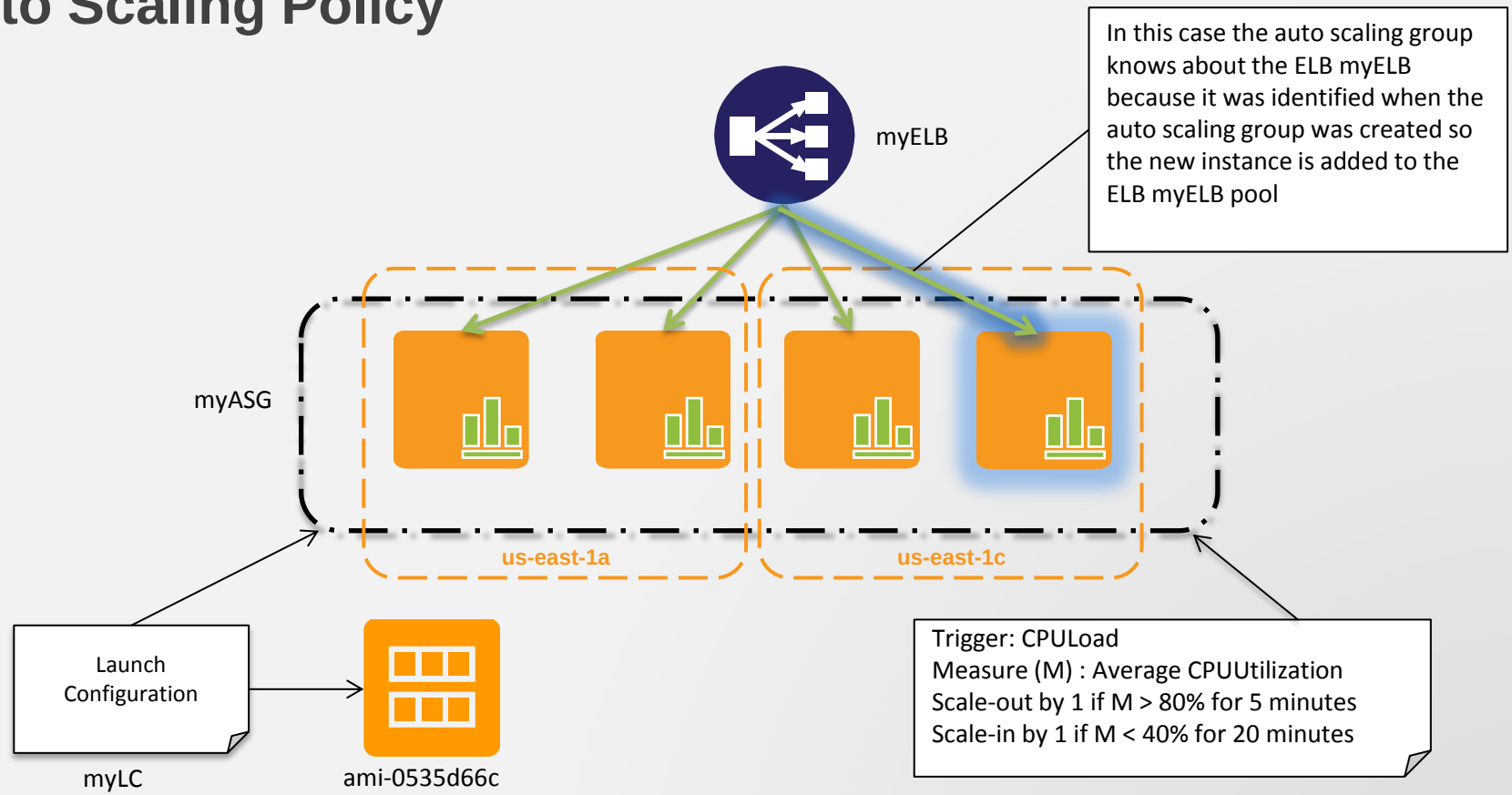
- Parameters for performing an Auto Scaling action
  - Scale Up/Down
  - By how much
    - ChangeInCapacity (+/- #)
    - ExactCapacity (#)
    - ChangeInPercent (+/- %)
  - Cool Down (seconds)
- Policy can be triggered by CloudWatch Events

Trigger: CPULoad
Measure (M) : Average CPUUtilization
Scale-out by 1 if M > 80% for 5 minutes
Scale-in by 1 if M < 40% for 20 minutes

Auto Scaling Policy



Auto Scaling Policy



Architecting with AWS

Elasticity, Scalability, and Bootstrapping

Appendix: Bootstrapping

Bootstrapping

- Bootstrapping is the process of automatically setting up your servers after they boot
- Auto scaling strategies must include proper bootstrapping of provisioned EC2 instances
    - AMI management
    - Software to install or configure (including rebooting)
    - Discovery or registration of new instances
- Low touch and as dynamic as possible is needed to meet high availability SLAs
- Graceful departure is just as important

Bootstrapping in the Real World

A MySQL read replica fails (unresponsive, too slow, etc.)

- First, you have to detect it
  - Is the instance running at all (i.e., pingable)?
  - Issue a standard “Show Slave Status” MySQL command against the Read Replica and look at “Seconds_Behind_Master”
  - “Show Slave Status” is also published as an Amazon CloudWatch metric (“Replica Lag”) available via the AWS Management Console or Amazon Cloud Watch APIs
- CloudWatch Alarm to SNS, Monitoring Agents and a monitoring console, or a witness instance are some ways to detect a bad read replica
- After its found, a bad read replica can be deleted and replaced

Bootstrapping in the Real World

So, the bad read replica is detected and replaced, but how does an app tier instance know about the new read replica?

- You can delete the bad read replica and create a new one with the same endpoint by using the same DB Instance Identifier and Source DB Instance Identifier as the deleted read replica. Automated or Manual through the AWS Console.
- If you deleted the bad read replica and replaced it with a new one with a different DB Instance Identifier and Source DB Instance Identifier
  - Automated - Something has to call the DescribeDBInstance API to retrieve the endpoint for the new read replica, then update the app tier instances(s) with the endpoint of the new read replica
  - Manual – Use the AWS Management Console to retrieve the endpoint for the new read replica and update the app tier