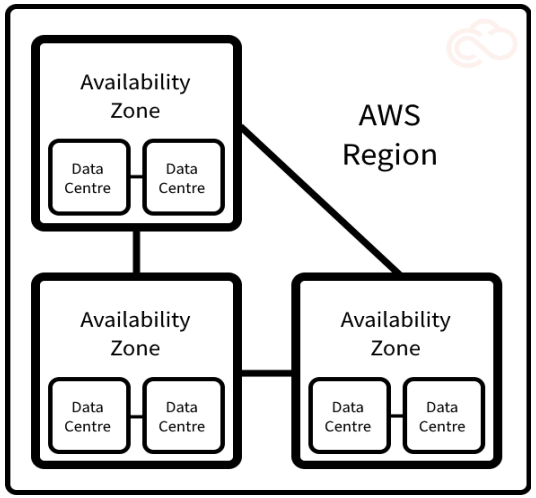
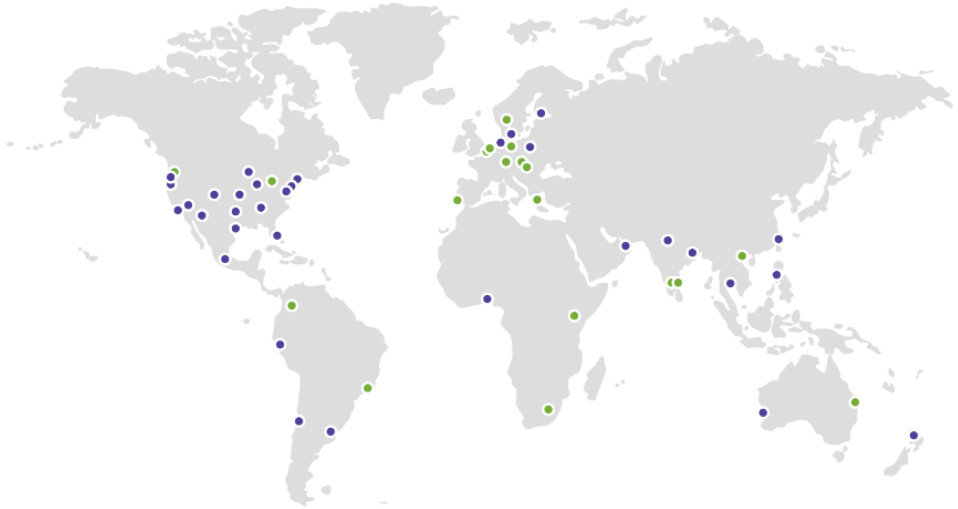
Lesson 1 Introduction

Michael Yang

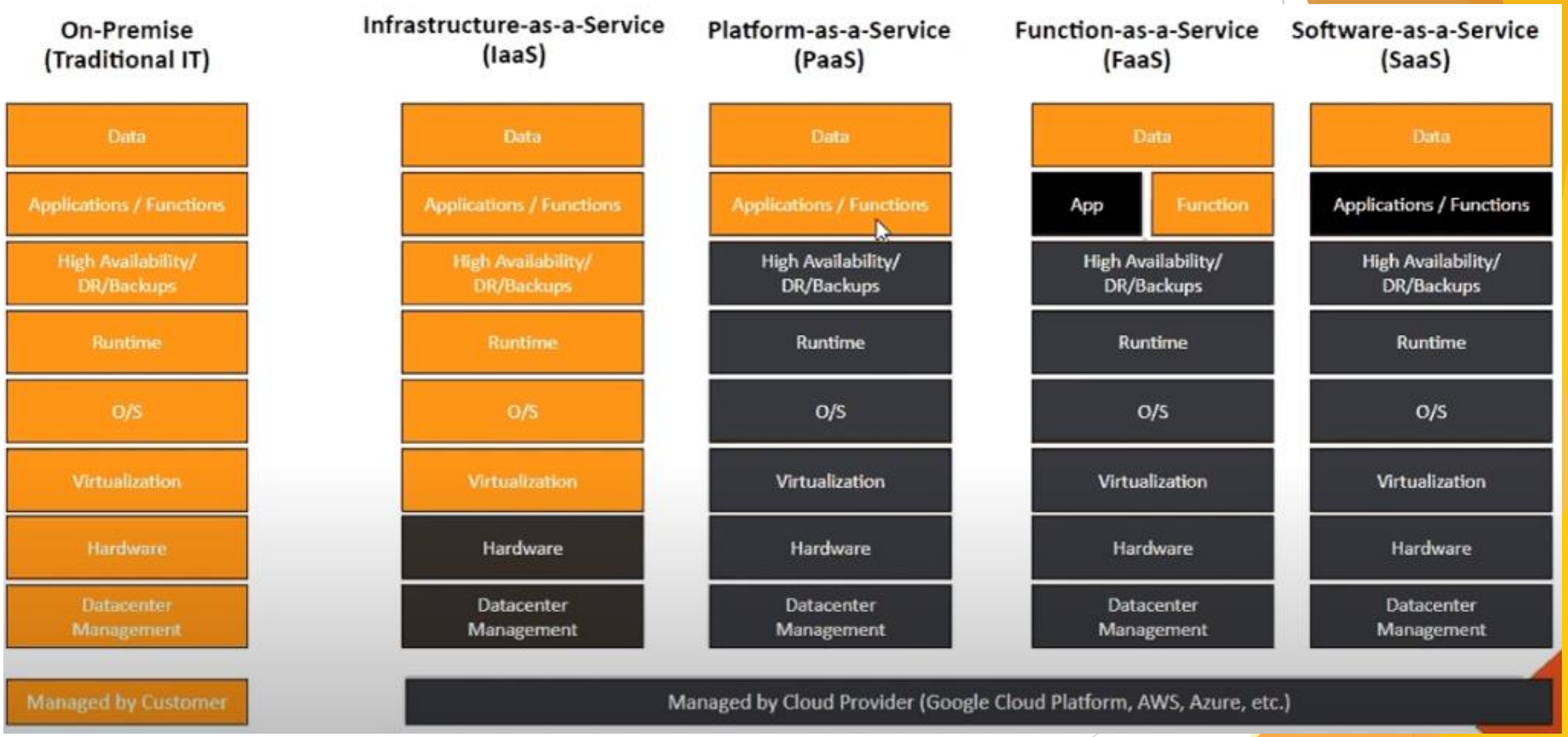
What is AWS

- *Amazon Web Services* (AWS) is a platform of web services that offers solutions for computing, storing, and networking, at different layers of abstraction.
- *Web services* are accessible via the internet by using typical web protocols (such as HTTP) and used by machines or by humans through a UI.
- As an AWS customer, you can choose among different *data centers*. AWS data centers are distributed worldwide.



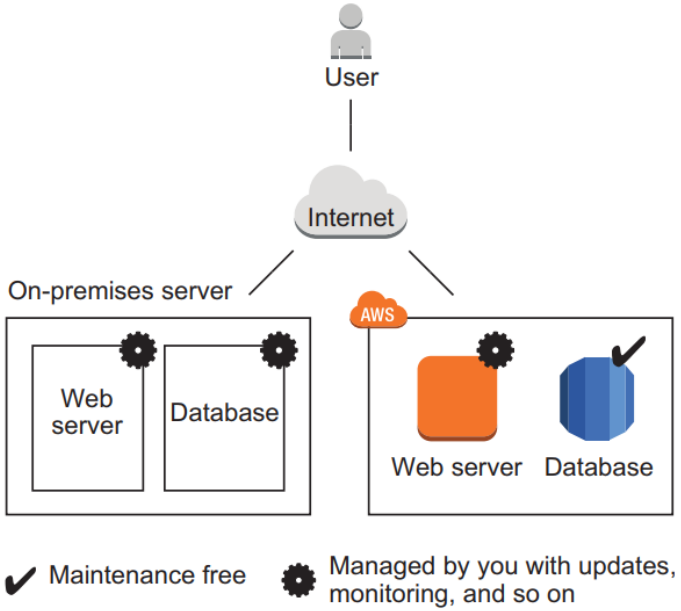


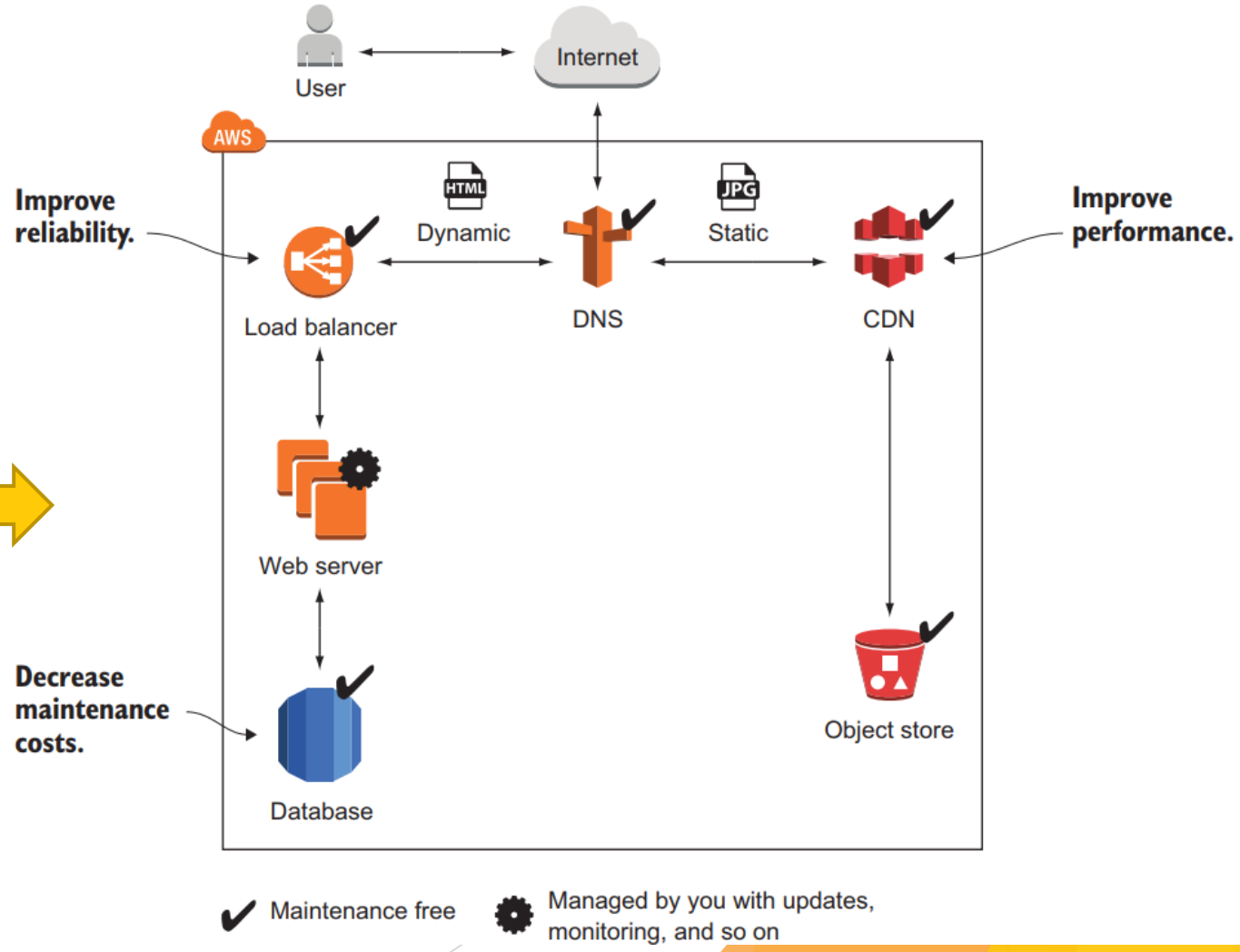
What is cloud computing



What can AWS do

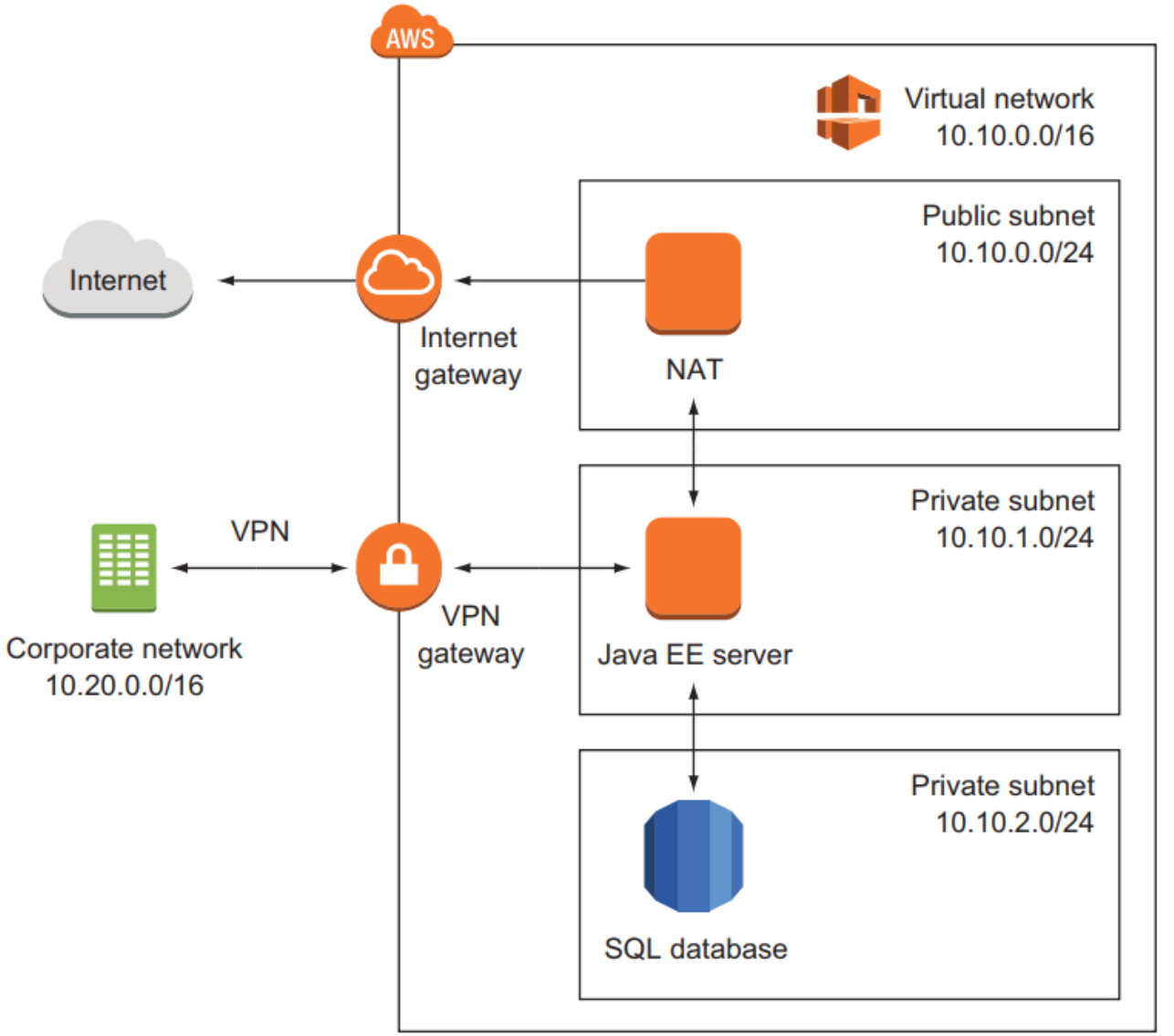
- Hosting a website





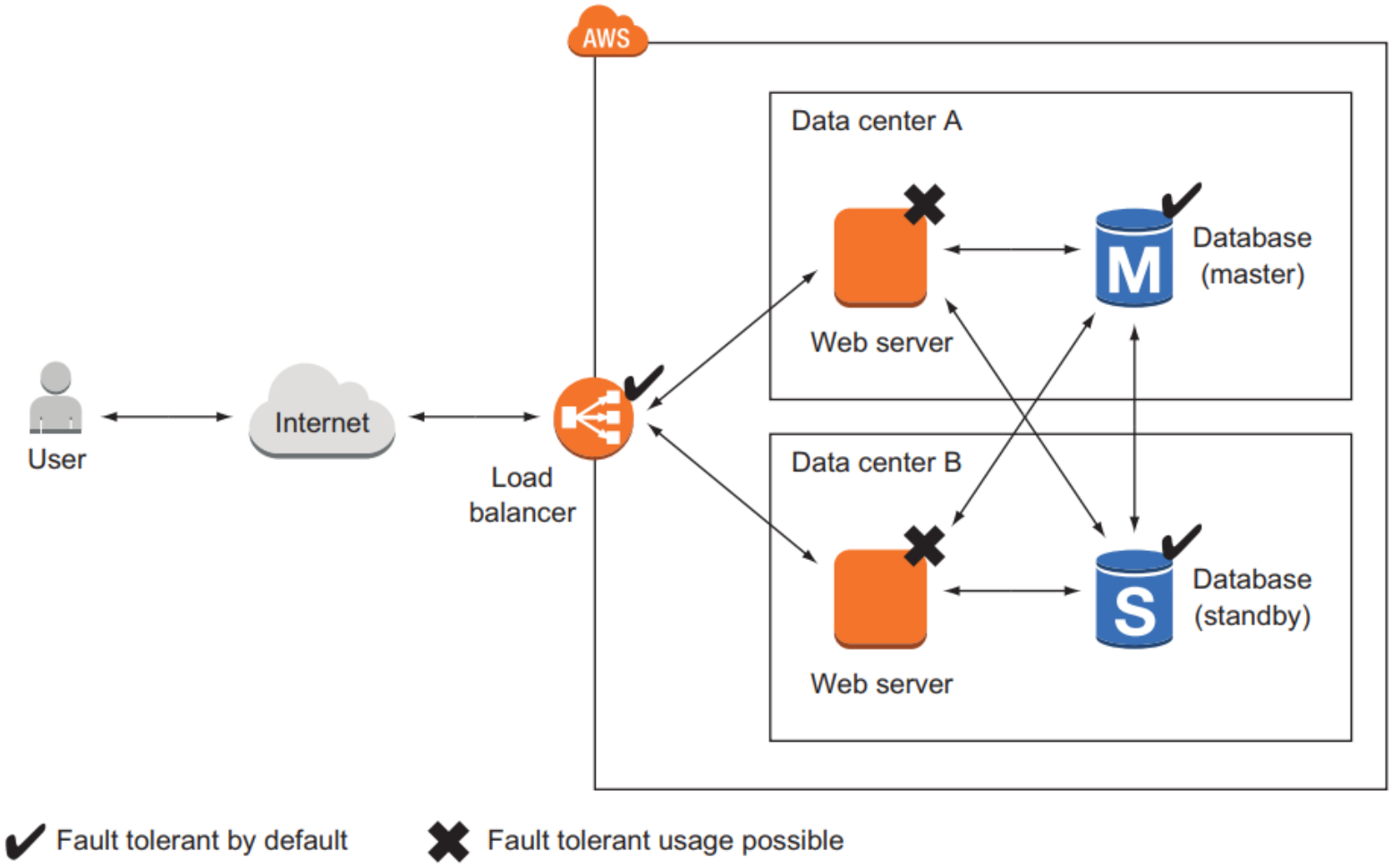
What can AWS do

- Hosting a Java EE application



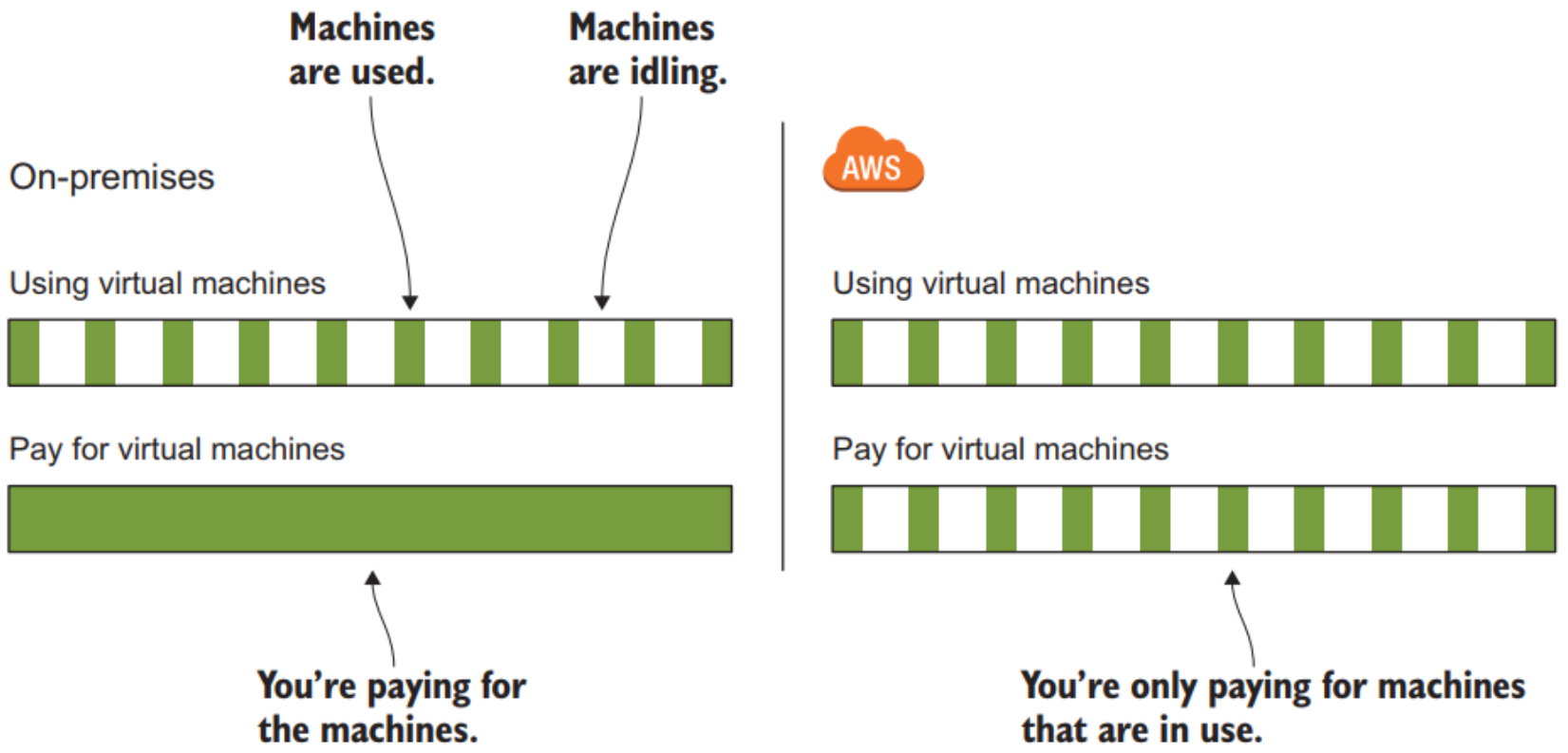
What can AWS do

- Implementing a highly available system



What can AWS do

- Profitting from low cost for batch processing infrastructure

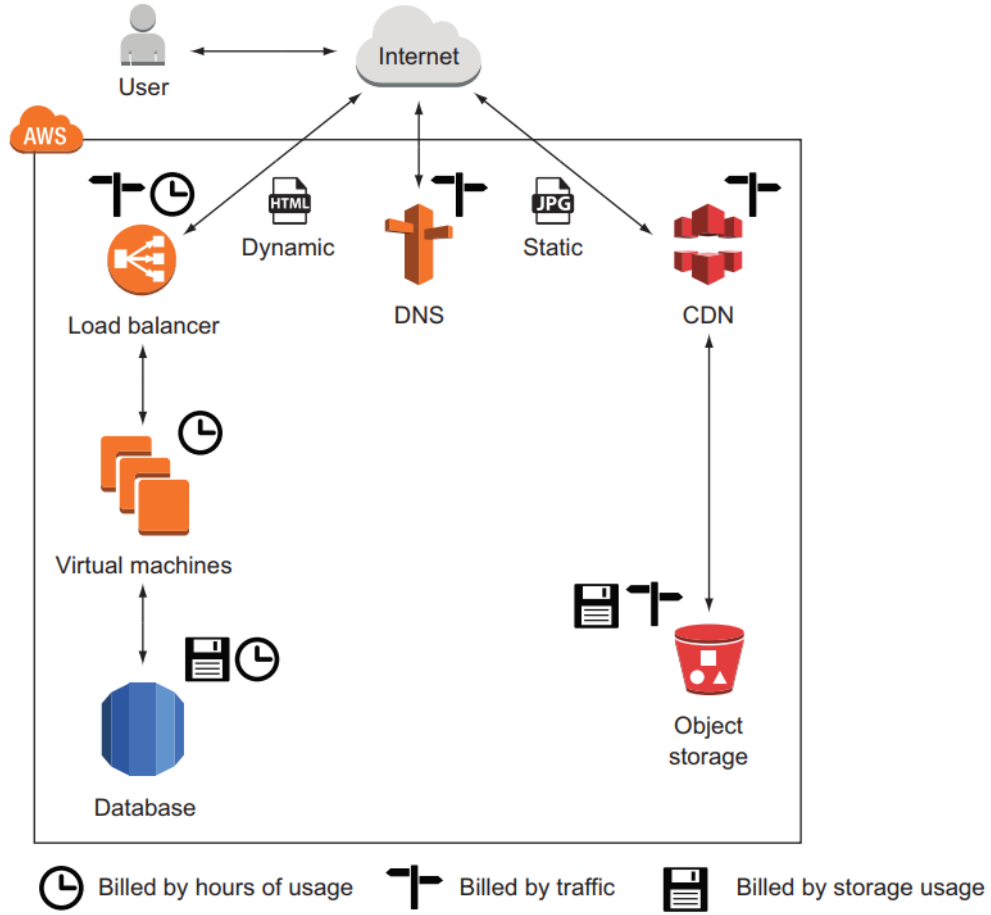


AWS Benefits

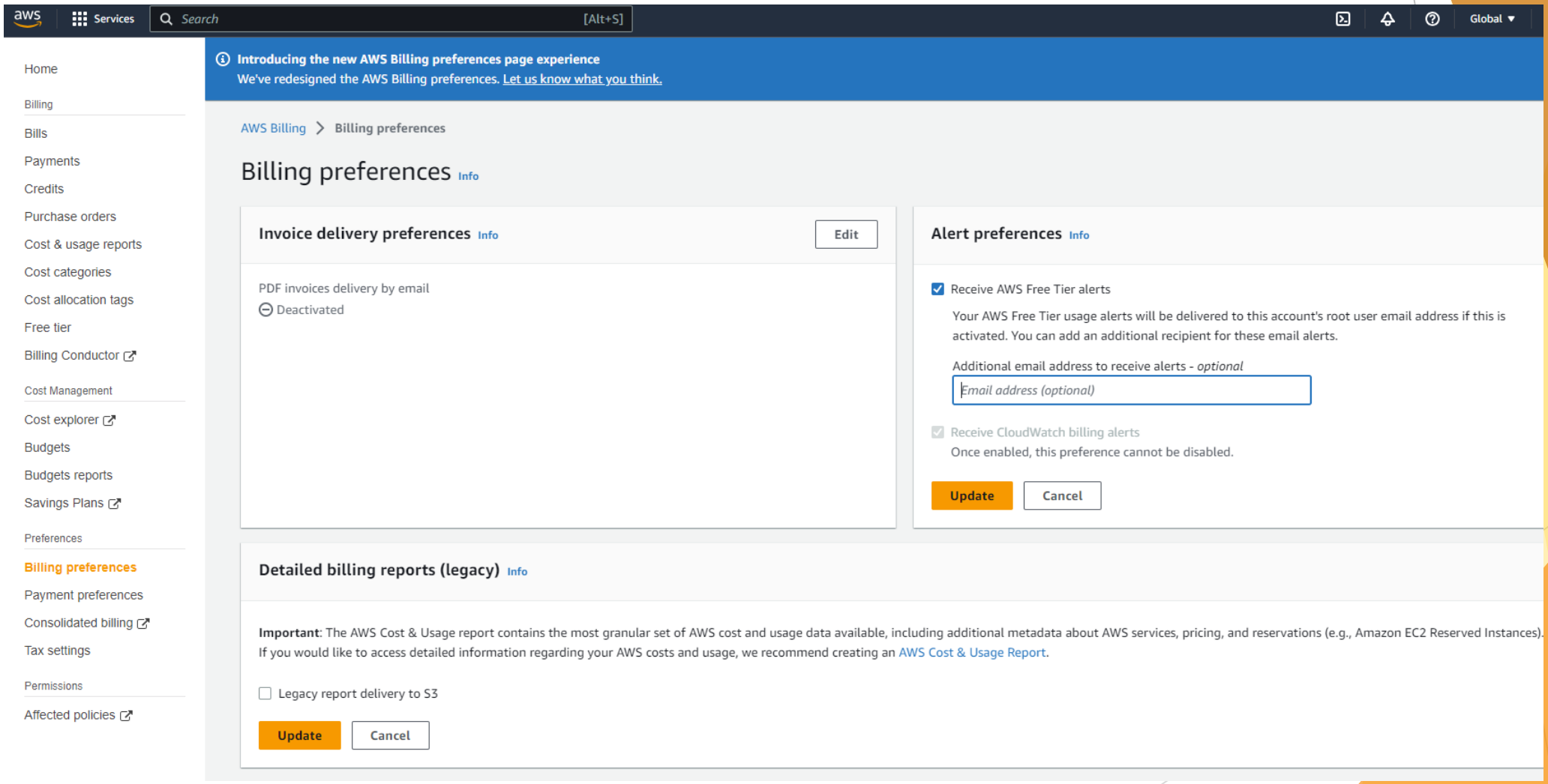
- Services solve common problems
- Enabling Automation
- Scalability (flexible capacity)
- Reliability (built for failure)
- Reducing time to market
- Global infrastructure

Billing

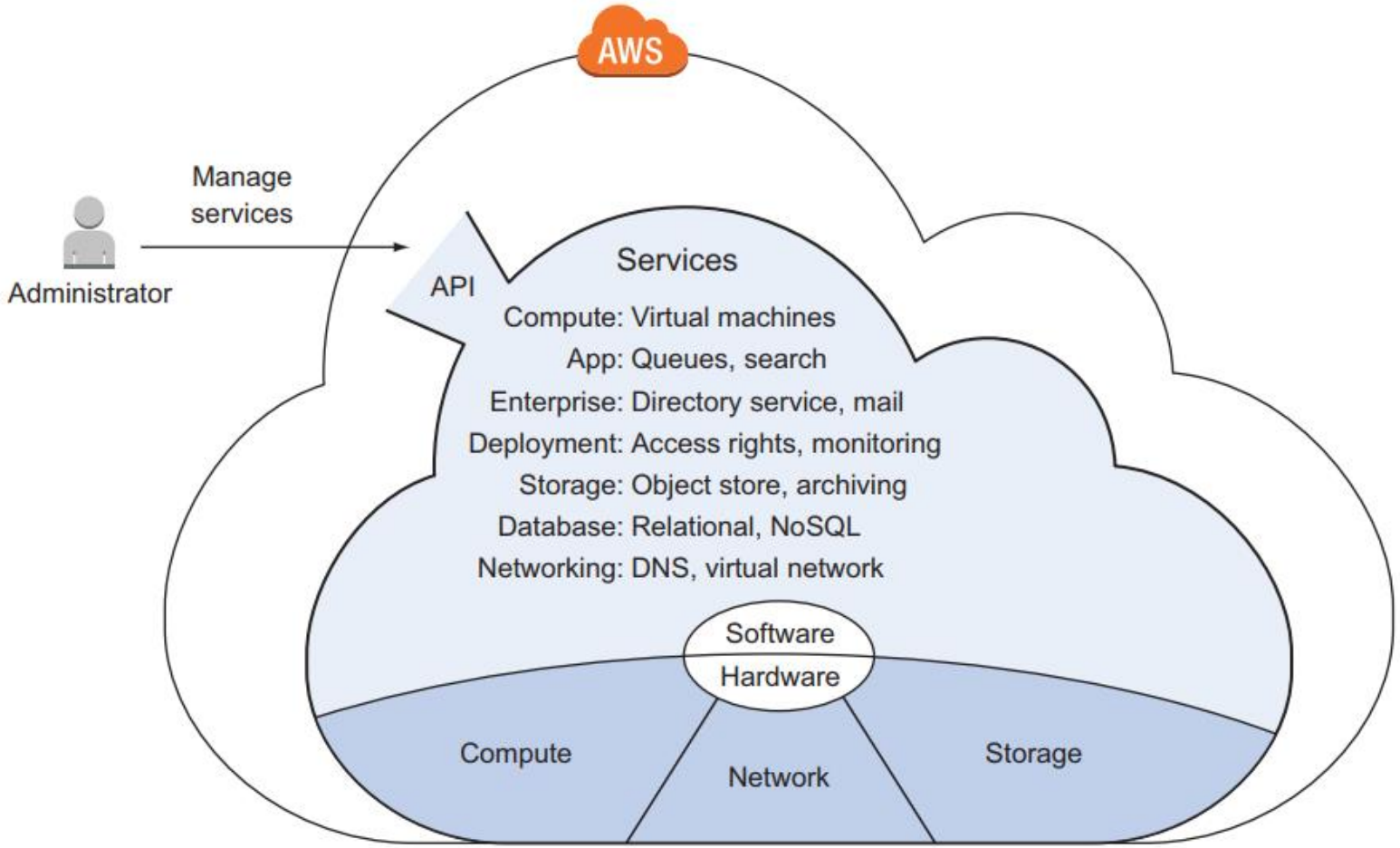
- Free Tier - You can use some AWS services for free within the first 12 months of your signing up
- Billing example



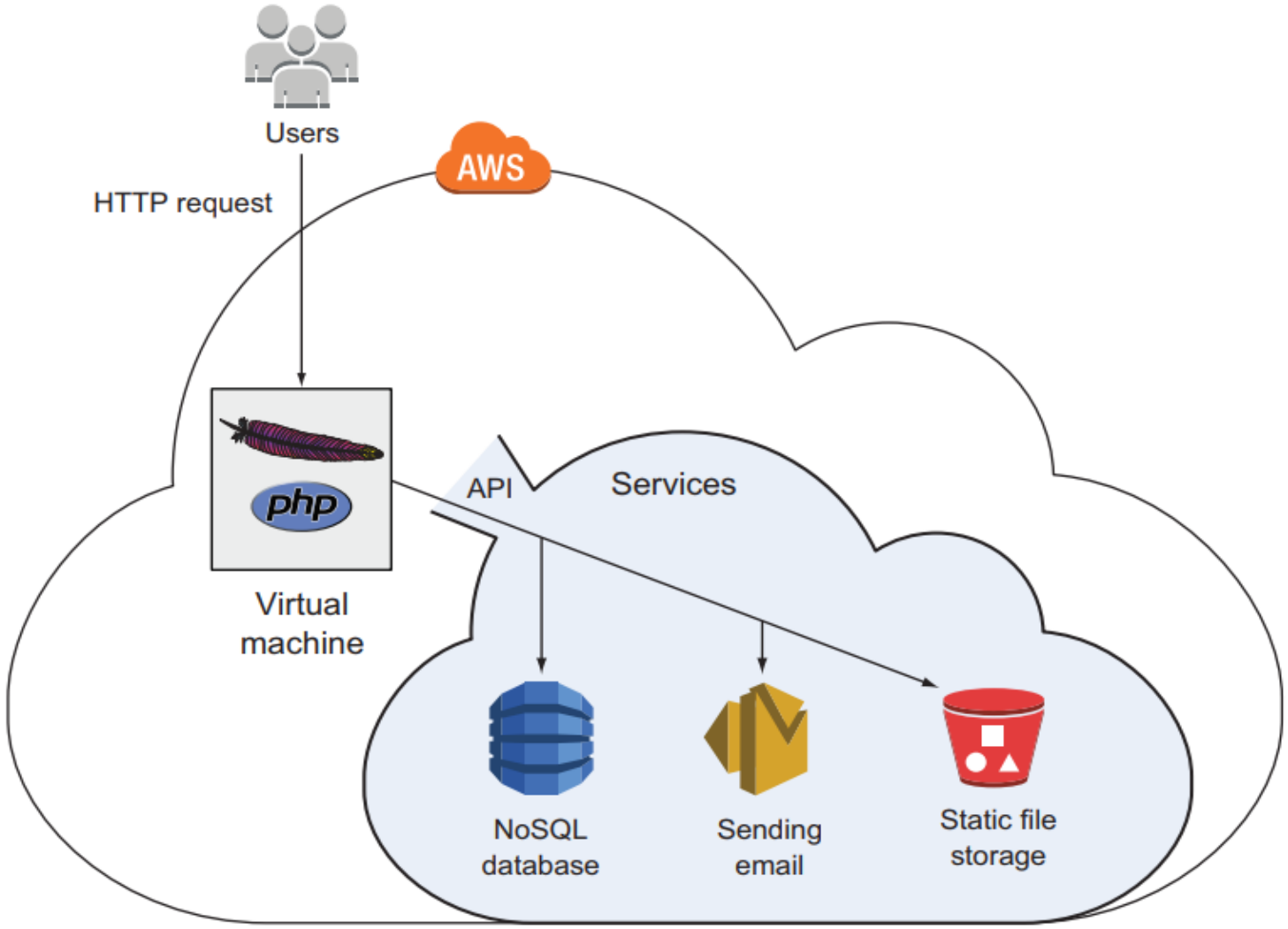
Track your Billing



Exploring AWS



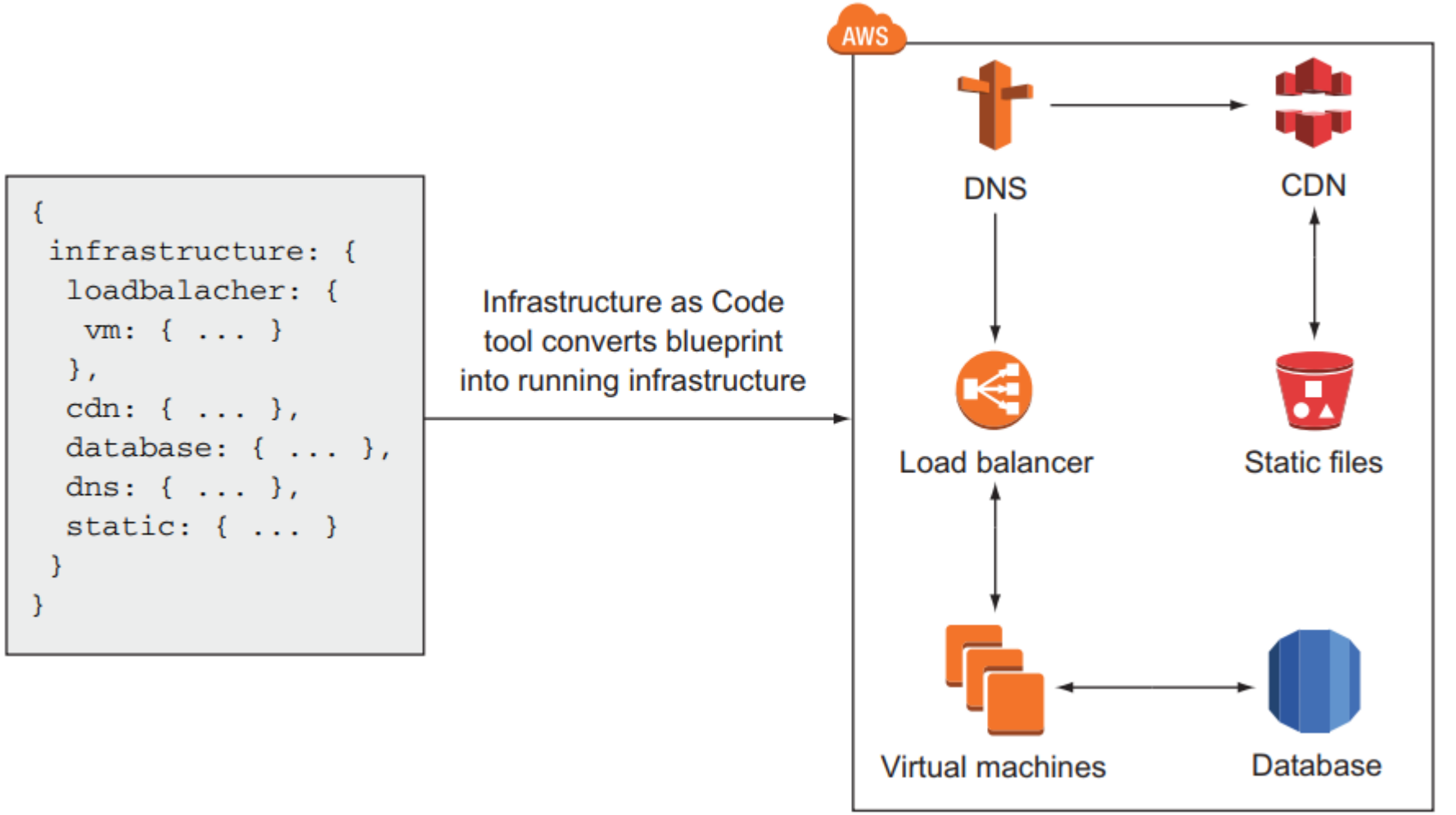
Exploring AWS



AWS Services Example

EC2—Virtual machines
ELB—Load balancers
Lambda—Executing functions
Elastic Beanstalk—Deploying web applications
S3—Object store
EFS—Network filesystem
Glacier—Archiving data
RDS—SQL databases
DynamoDB—NoSQL database
ElastiCache—In-memory key-value store
VPC—Private network
CloudWatch—Monitoring and logging
CloudFormation—Automating your infrastructure
OpsWorks—Deploying web applications
IAM—Restricting access to your cloud resources
Simple Queue Service—Distributed queues

Interacting with AWS



AWS Account

- **Container** – An AWS account is the basic container for all the AWS resources you create as an AWS customer.
- **Security boundary** – An AWS account is also the basic security boundary for your AWS resources. Resources that you create in your account are available to users who have credentials for your account.

Among the key resources you can create in your account are ***identities***, such as ***users*** and ***roles***. Identities have credentials that someone can use to sign in (*authenticate*) to AWS. Identities also have permission policies that specify what a user can do (*authorization*) with the resources in the account.

Exercise 1. Signing up

- The sign-up process consists of five steps:

1 Providing login credentials
2 Providing contact information
3 Providing payment details
4 Verifying your identity
5 Choosing a support plan

Root user vs IAM user

- The root user is the account owner and is created when the AWS account is created. Other types of users, including IAM users, and AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center) users are created by the root user or an administrator for the account. All AWS users have security credentials.
- **Root User credentials**

The credentials of the account owner allow full access to all resources in the account. You can't use IAM policies to explicitly deny the root user access to resources.

- **IAM credentials**

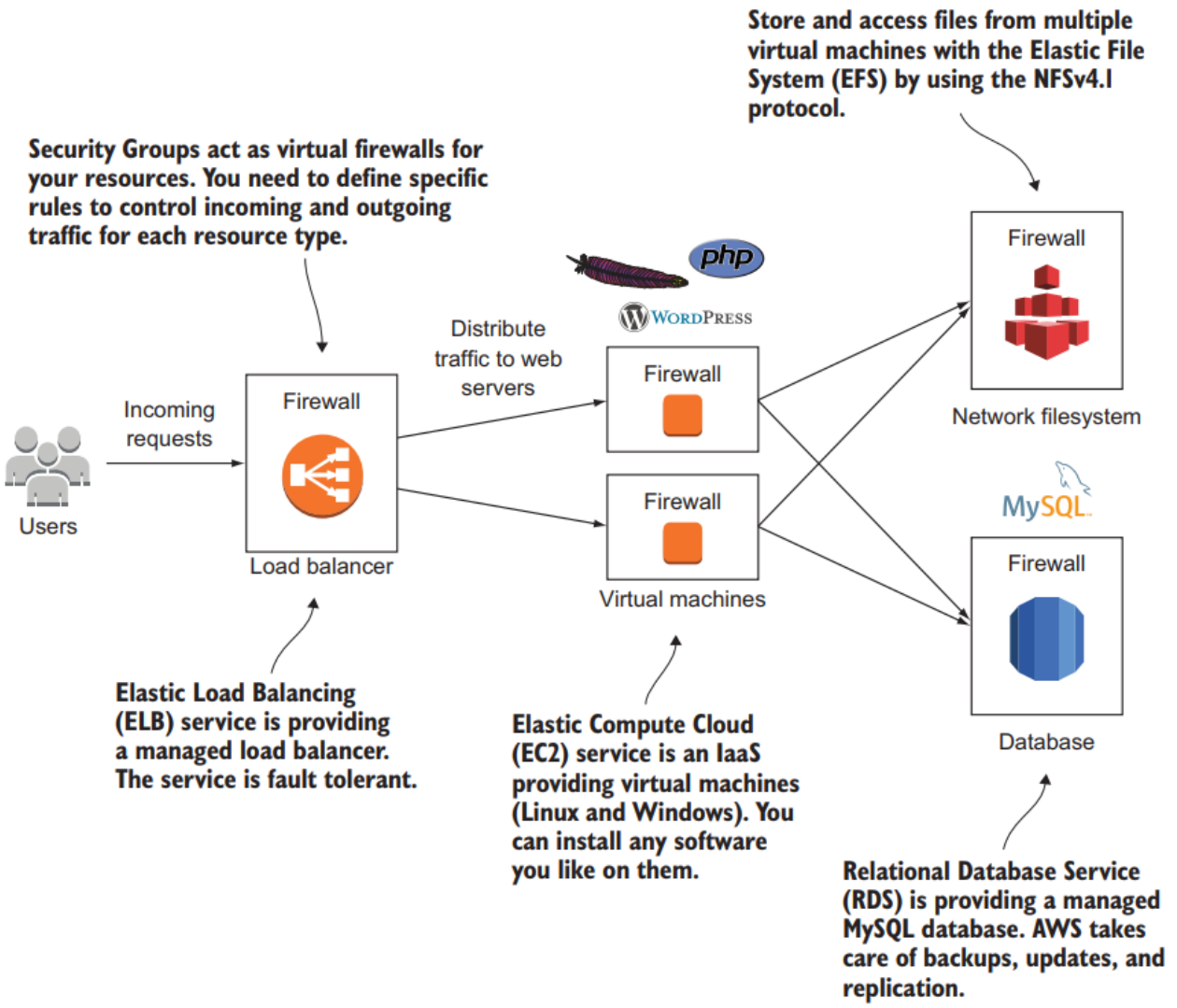
An IAM user is an entity you create in AWS that represents the person or service that uses the IAM user to interact with AWS resources. These users are identities within your AWS account that have specific custom permissions.

Access Keys

- When you use AWS programmatically, you provide your AWS access keys so that AWS can verify your identity in programmatic calls. Your access keys consist of an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY).

Exercise 2 Exploring AWS

- Deploy an application *wordexpress* on AWS
- Infrastructure as Code



Summary

- Amazon Web Services (AWS) is a platform of web services for computing, storing, and networking that work well together.
- Cost savings aren't the only benefit of using AWS. You'll also profit from an innovative and fast-growing platform with flexible capacity, fault-tolerant services, and a worldwide infrastructure.
- Any use case can be implemented on AWS, whether it's a widely used web application or a specialized enterprise application with an advanced networking setup.
- You can interact with AWS in many different ways. You can control the different services by using the web-based GUI, use code to manage AWS programmatically from the command line or SDKs, or use blueprints to set up, modify, or delete your infrastructure on AWS.
- Pay-per-use is the pricing model for AWS services. Computing power, storage, and networking services are billed similarly to electricity.
- Creating an AWS account is easy. Now you know how to set up a key pair so you can log in to virtual machines for later use.
- Creating a billing alarm allows you to keep track of your AWS bill and get notified whenever you exceed the Free Tier.

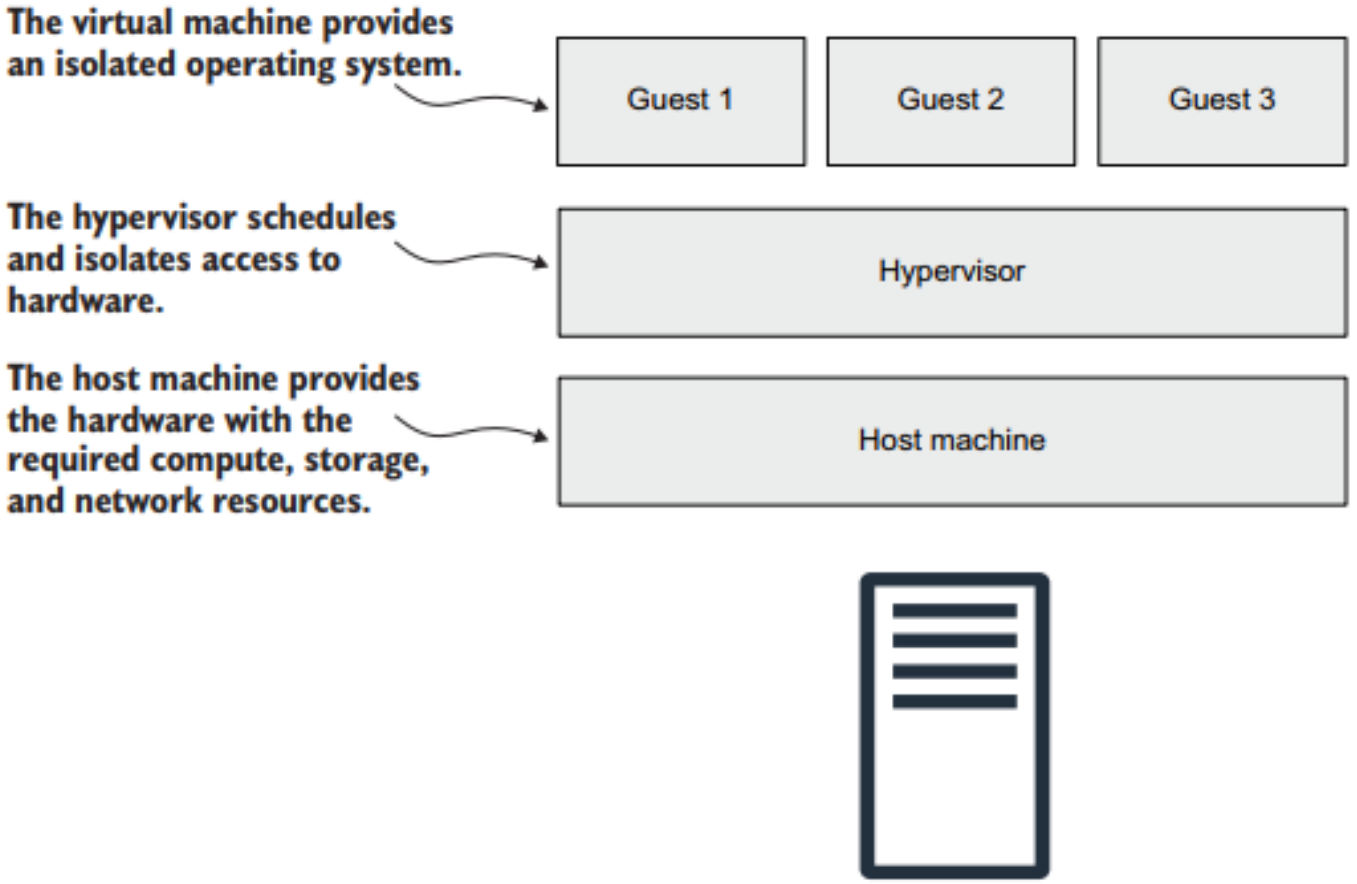
Lesson 2 EC2

Michael Yang

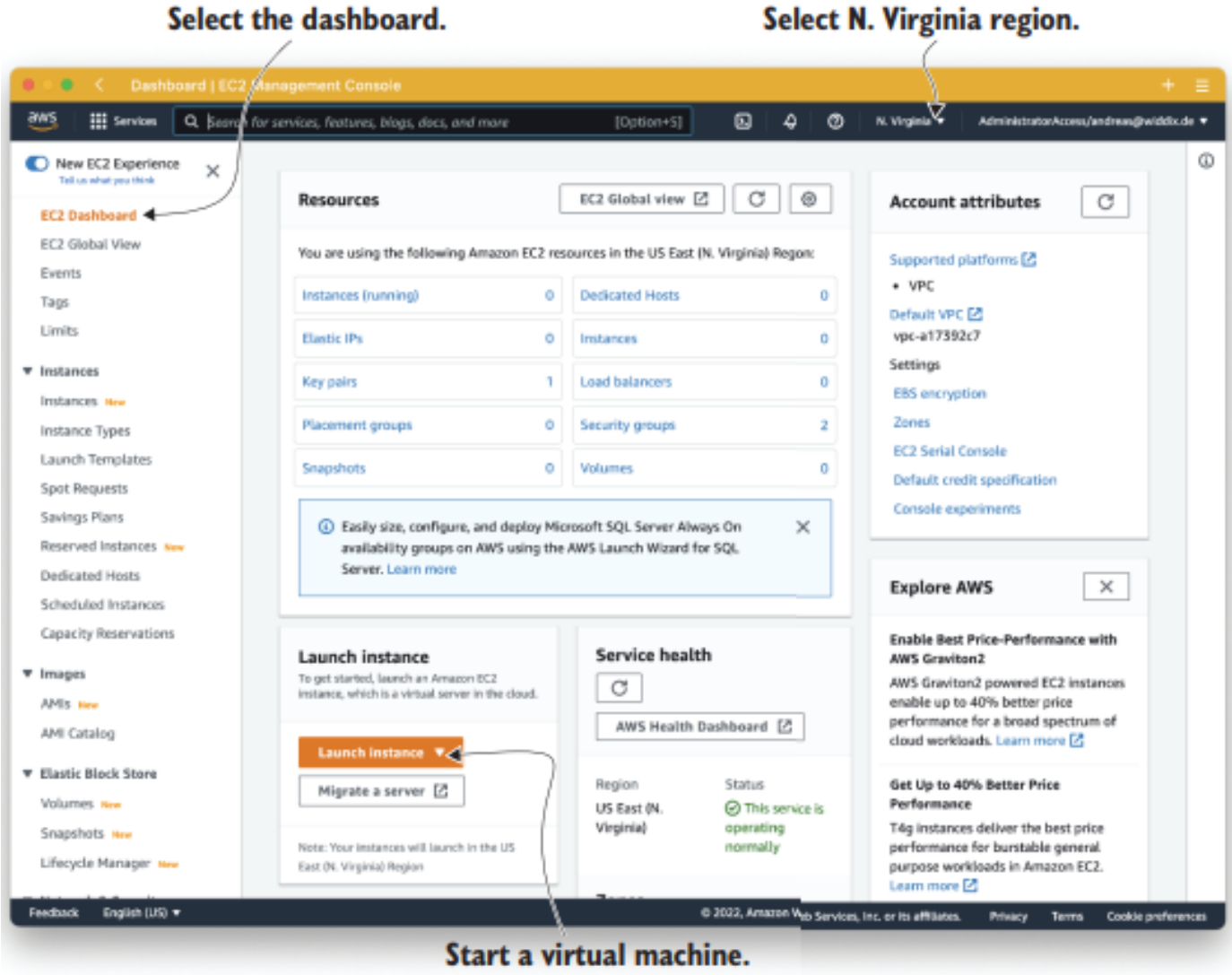
What is Virtual Machine(VM)

- With a virtual machine, you get access to a slice of a physical machine located in a data center.
- Typical use cases for a virtual machine follow:

- Hosting a web application such as WordPress
- Operating an enterprise application, such as an ERP application
- Transforming or analyzing data, such as encoding video files



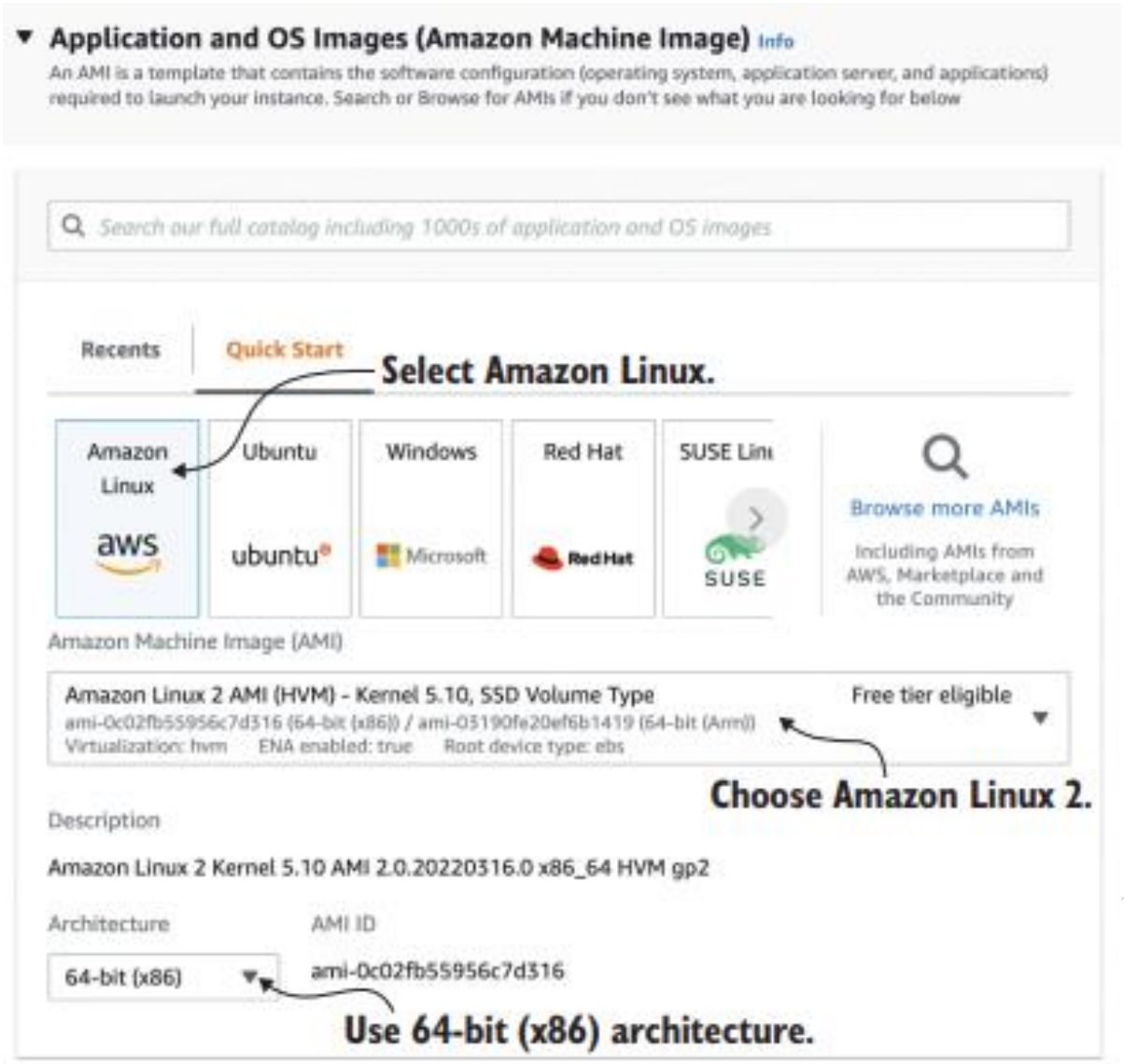
Launch a VM


Select the dashboard.
Select N. Virginia region.
Dashboard | EC2 Management Console
Services
Search for services, features, blogs, docs, and more [Option+S]
N. Virginia
AdministratorAccess/andreas@widdix.de
New EC2 Experience
EC2 Dashboard
EC2 Global View
Events
Tags
Limits
Instances
Instances
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances
Dedicated Hosts
Scheduled Instances
Capacity Reservations
Images
AMIs
AMI Catalog
Elastic Block Store
Volumes
Snapshots
Lifecycle Manager
Resources
EC2 Global view
You are using the following Amazon EC2 resources in the US East (N. Virginia) Region:
Instances (running) 0
Dedicated Hosts 0
Elastic IPs 0
Instances 0
Key pairs 1
Load balancers 0
Placement groups 0
Security groups 2
Snapshots 0
Volumes 0
Easily size, configure, and deploy Microsoft SQL Server Always On availability groups on AWS using the AWS Launch Wizard for SQL Server. Learn more
Launch instance
To get started, launch an Amazon EC2 instance, which is a virtual server in the cloud.
Launch instance
Migrate a server
Note: Your instances will launch in the US East (N. Virginia) Region
Service health
AWS Health Dashboard
Region
US East (N. Virginia)
Status
This service is operating normally
Account attributes
Supported platforms
VPC
Default VPC
vpc-a17392c7
Settings
EBS encryption
Zones
EC2 Serial Console
Default credit specification
Console experiments
Explore AWS
Enable Best Price-Performance with AWS Graviton2
AWS Graviton2 powered EC2 instances enable up to 40% better price performance for a broad spectrum of cloud workloads. Learn more
Get Up to 40% Better Price Performance
T4g instances deliver the best price performance for burstable general purpose workloads in Amazon EC2. Learn more
Feedback
English (US)
© 2022, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Start a virtual machine.

Launch a VM Guideline

1. Naming the virtual machine
2. Selecting the operating system (OS)
3. Choosing the size of your virtual machine
4. Configuring details
5. Adding storage
6. Configuring a firewall
7. Granting the virtual machine permissions to access other AWS services

Choosing Amazon Machine Image - AMI

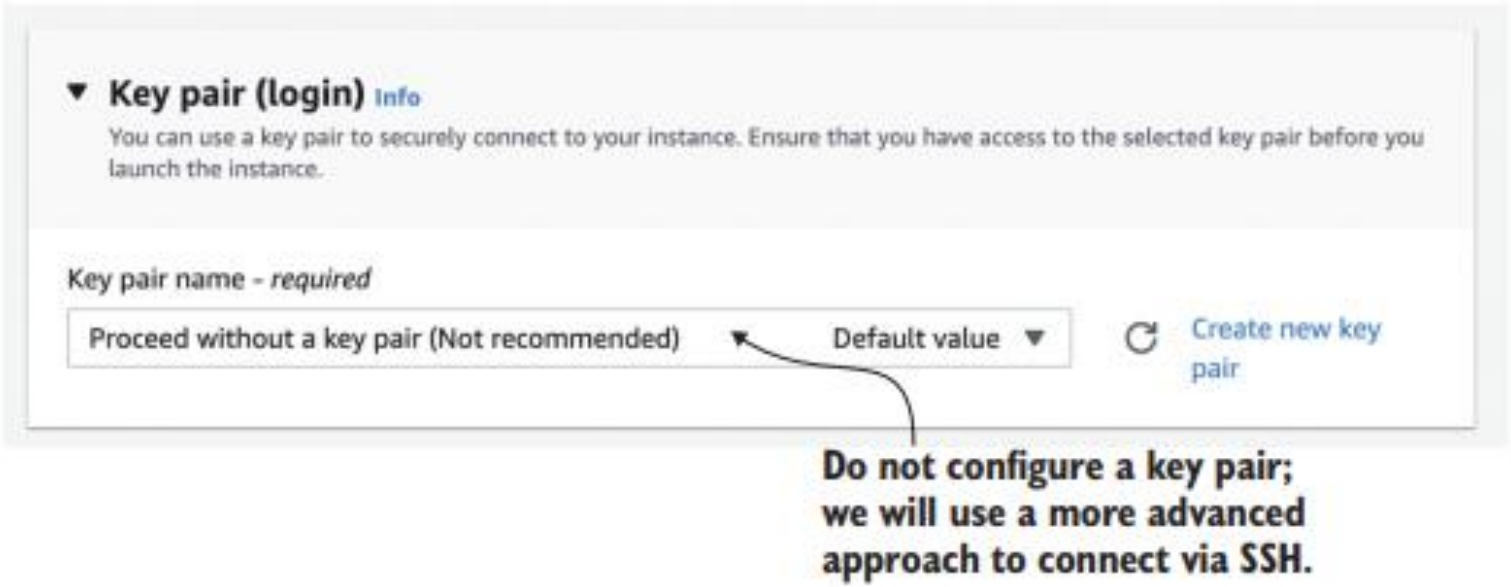


Choosing the size of VM

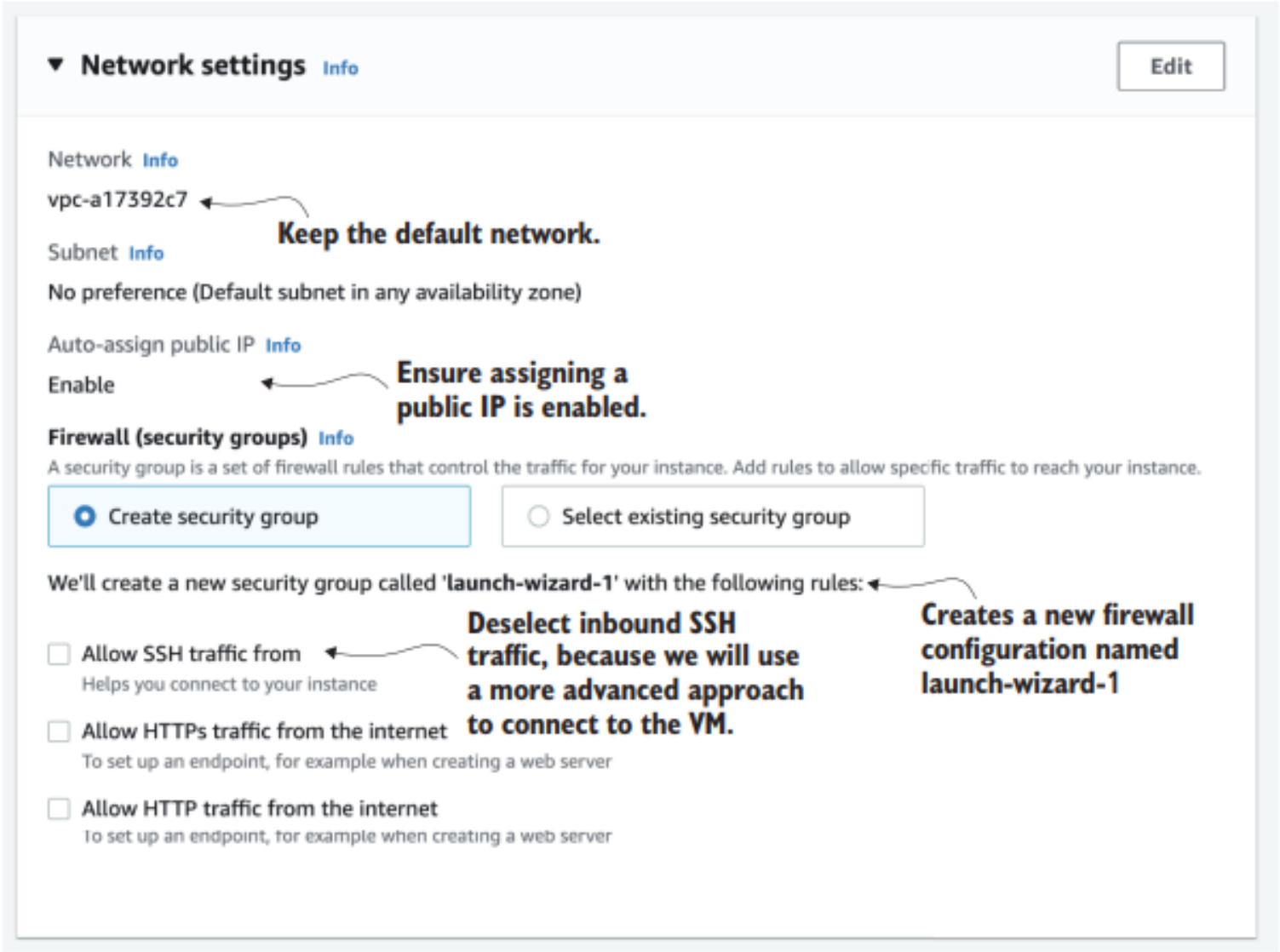
- https://aws.amazon.com/ec2/instance-types/

Instance type	Virtual CPUs	Memory	Description	Typical use case	Monthly cost (USD)
t2.nano	1	0.5 GiB	Small and cheap instance type, with moderate baseline performance and the ability to burst CPU performance above the baseline	Testing and development environments and applications with very low traffic	$4
m6i.large	2	8 GiB	Has a balanced ratio of CPU, memory, and networking performance	All kinds of applications, such as medium databases, web servers, and enterprise applications	$69
r6i.large	2	16 GiB	Optimized for memory-intensive applications with extra memory	In-memory caches and enterprise application servers	$90

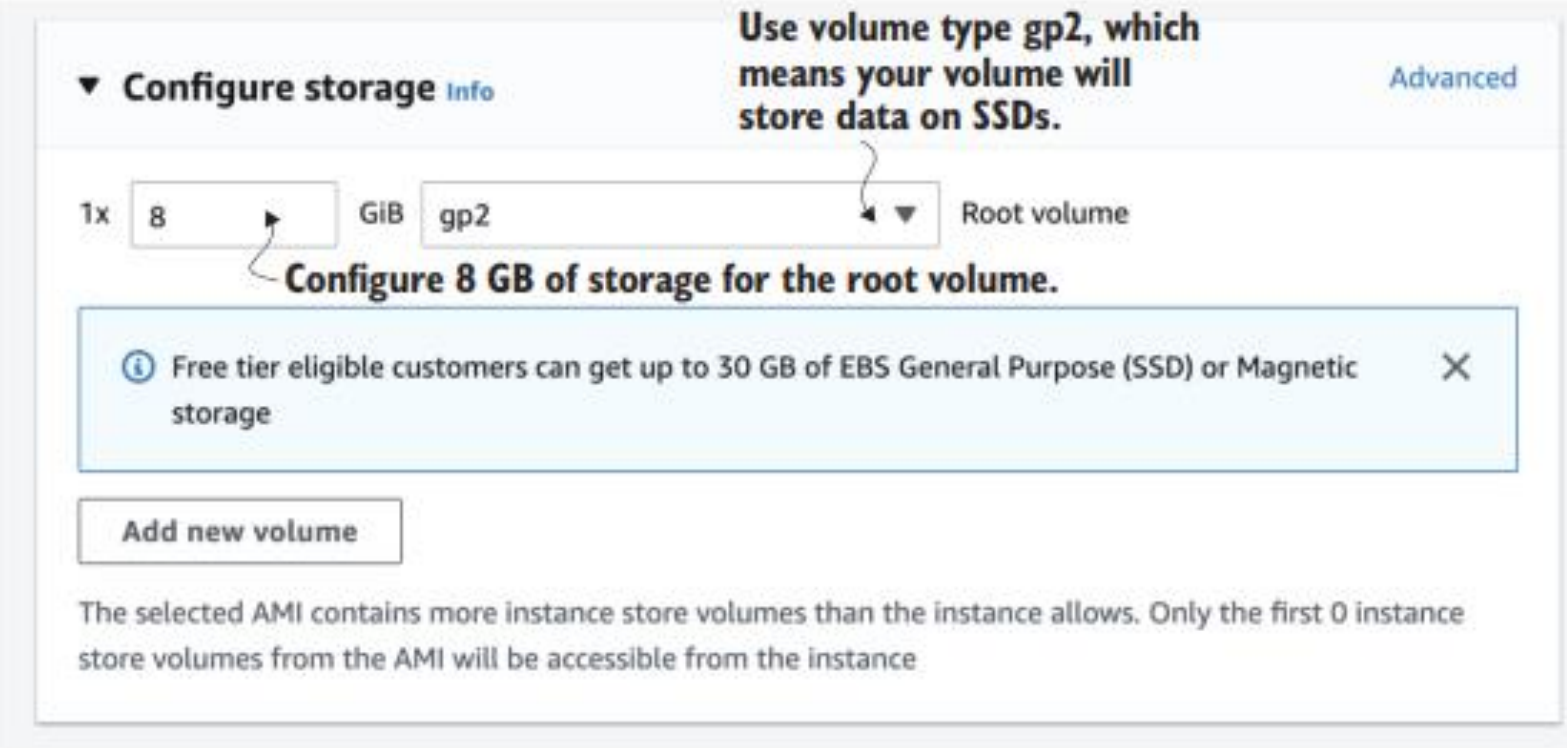
Configuring Key Pair



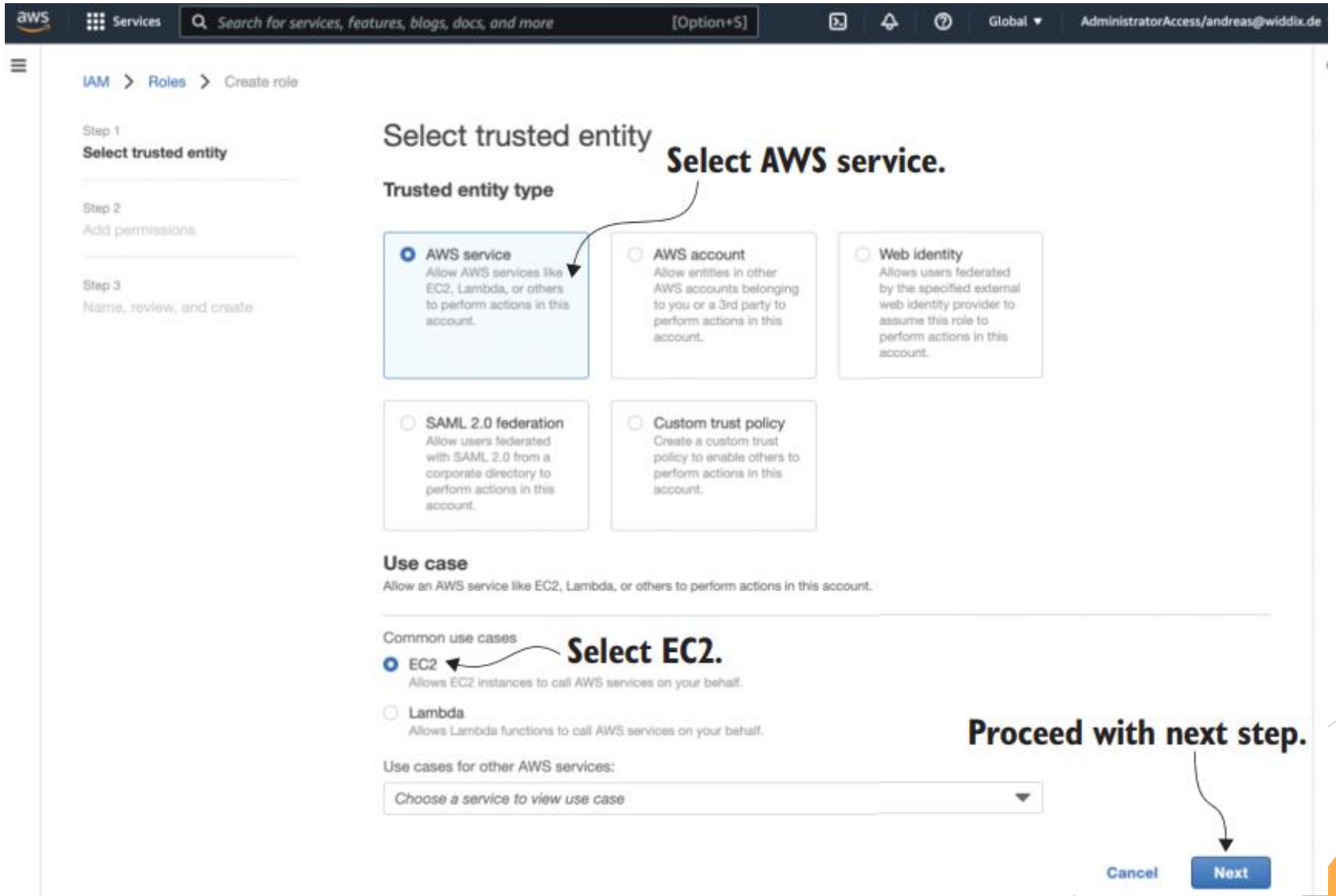
Network and Firewall Setting



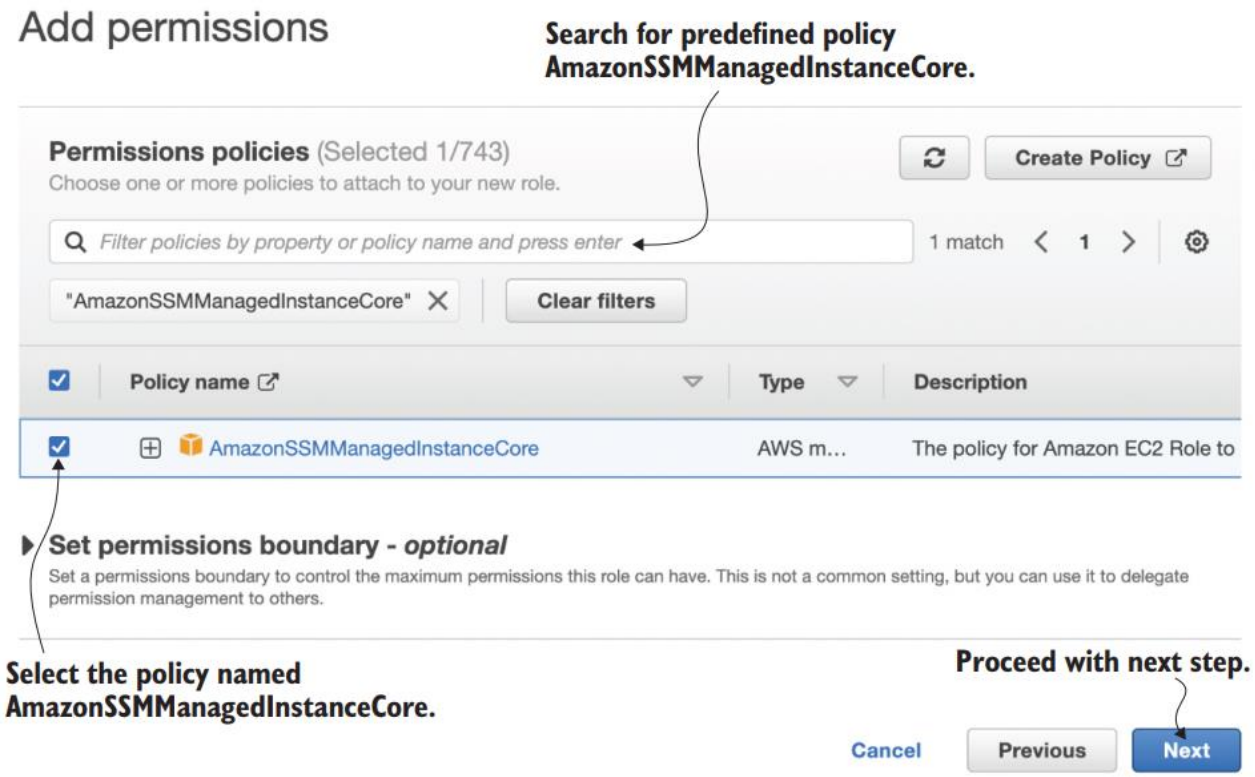
Configure storage

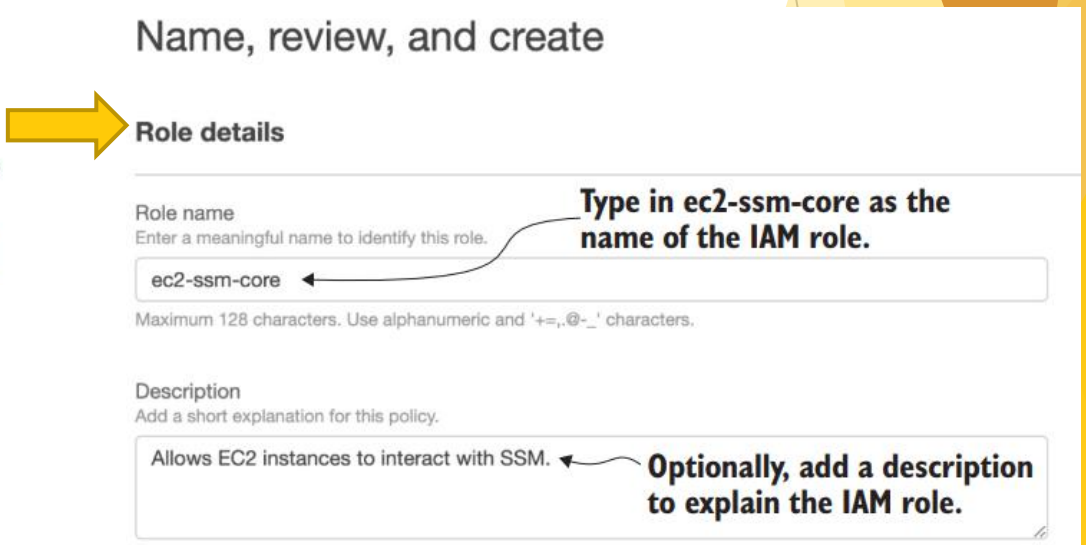


Creating an IAM role

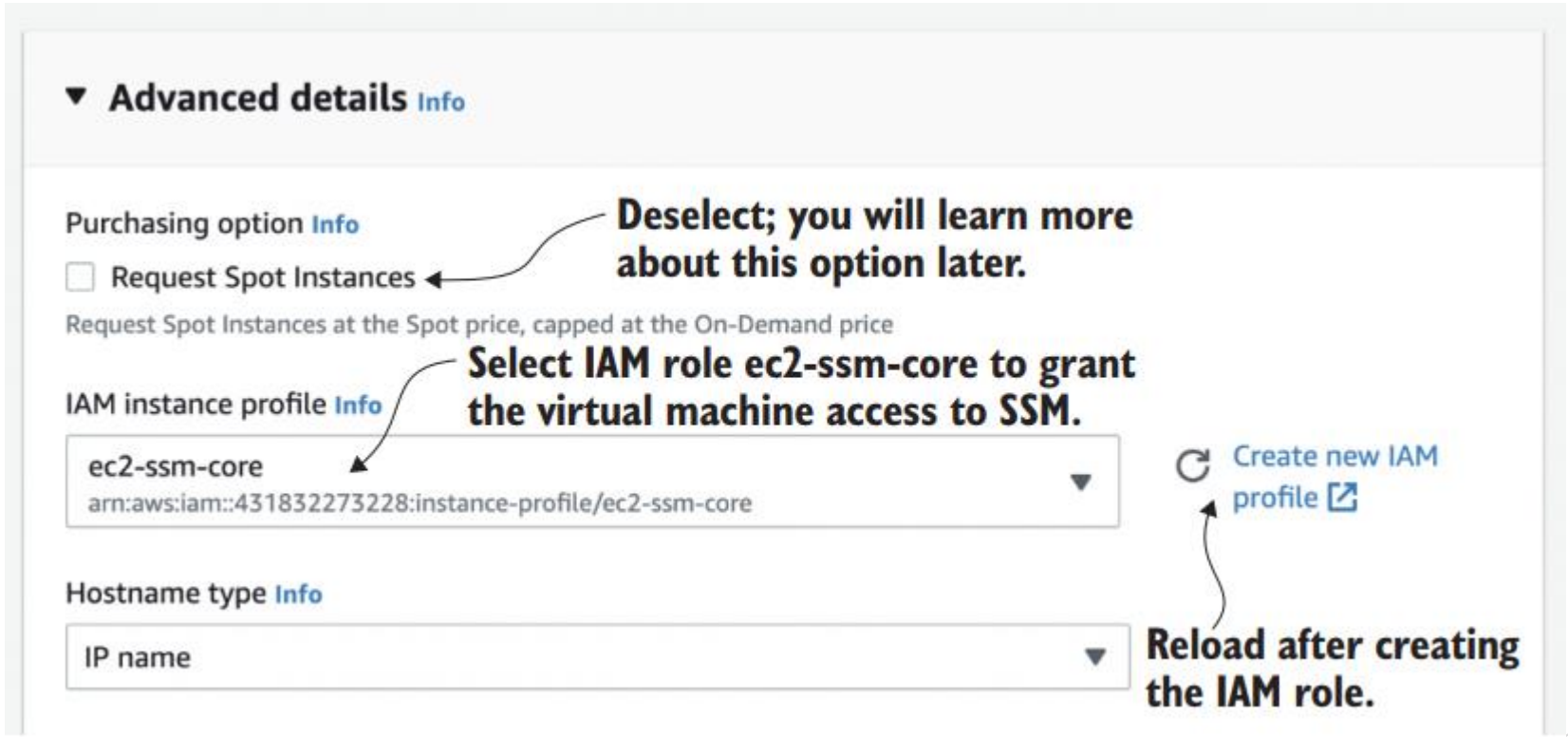


Creating an IAM role

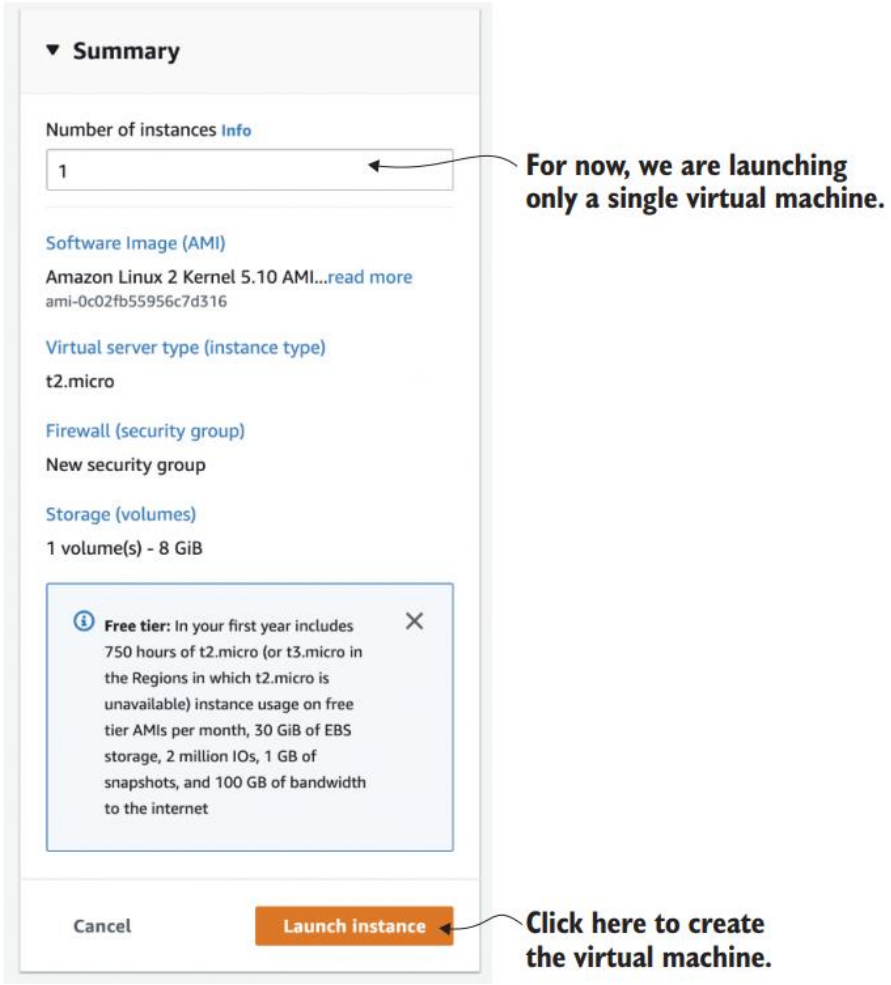




Configure IAM role for EC2 Instance



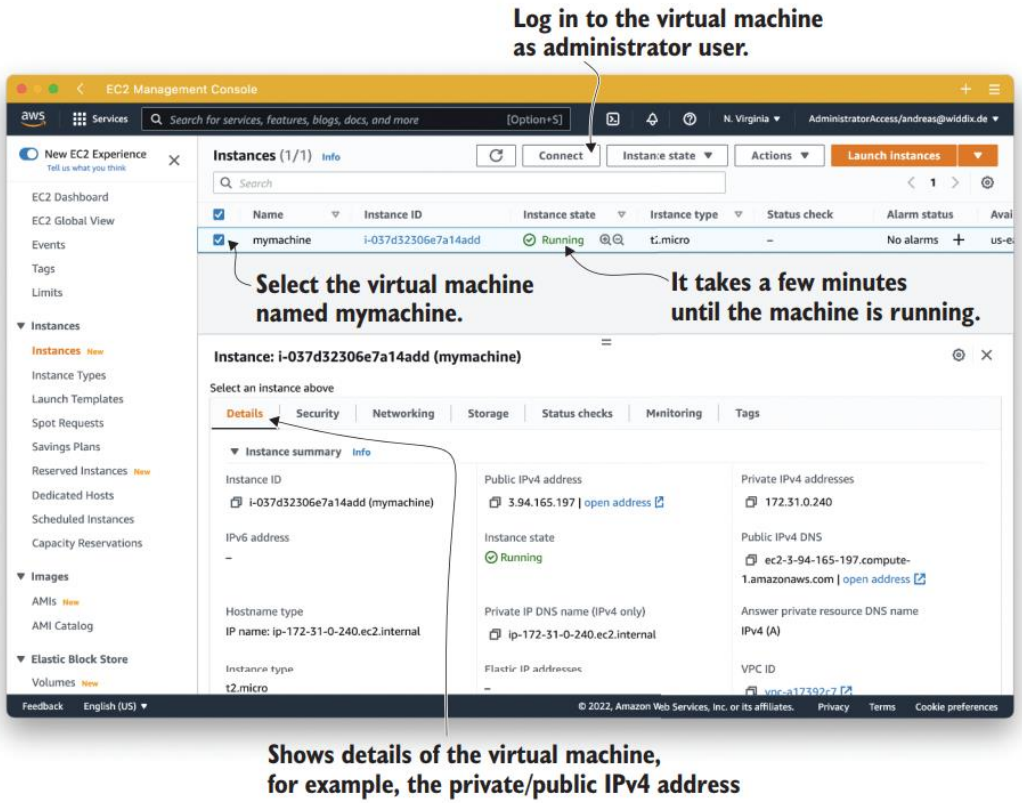
Launch a EC2 Instance

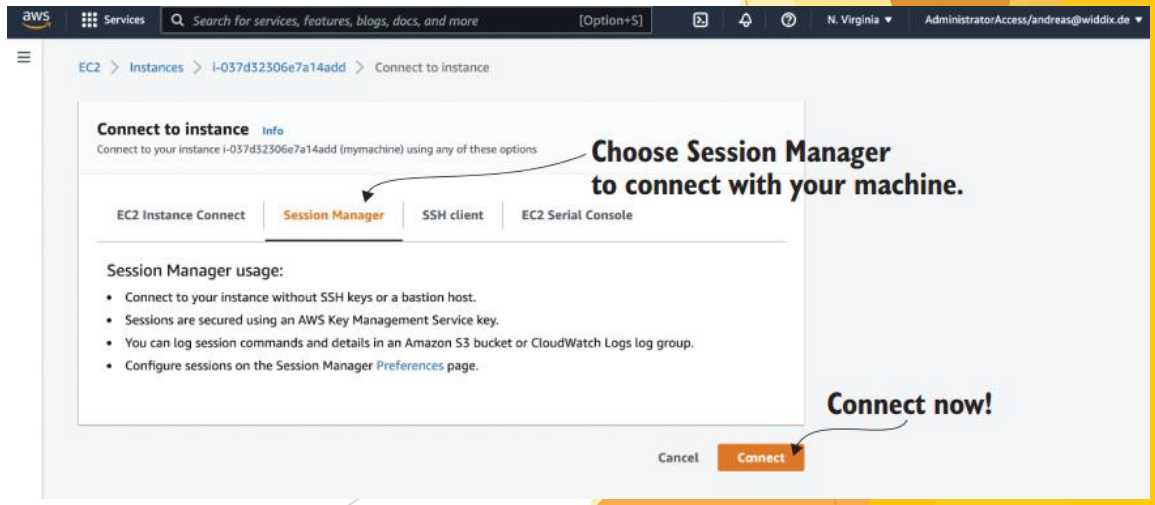


For now, we are launching only a single virtual machine.

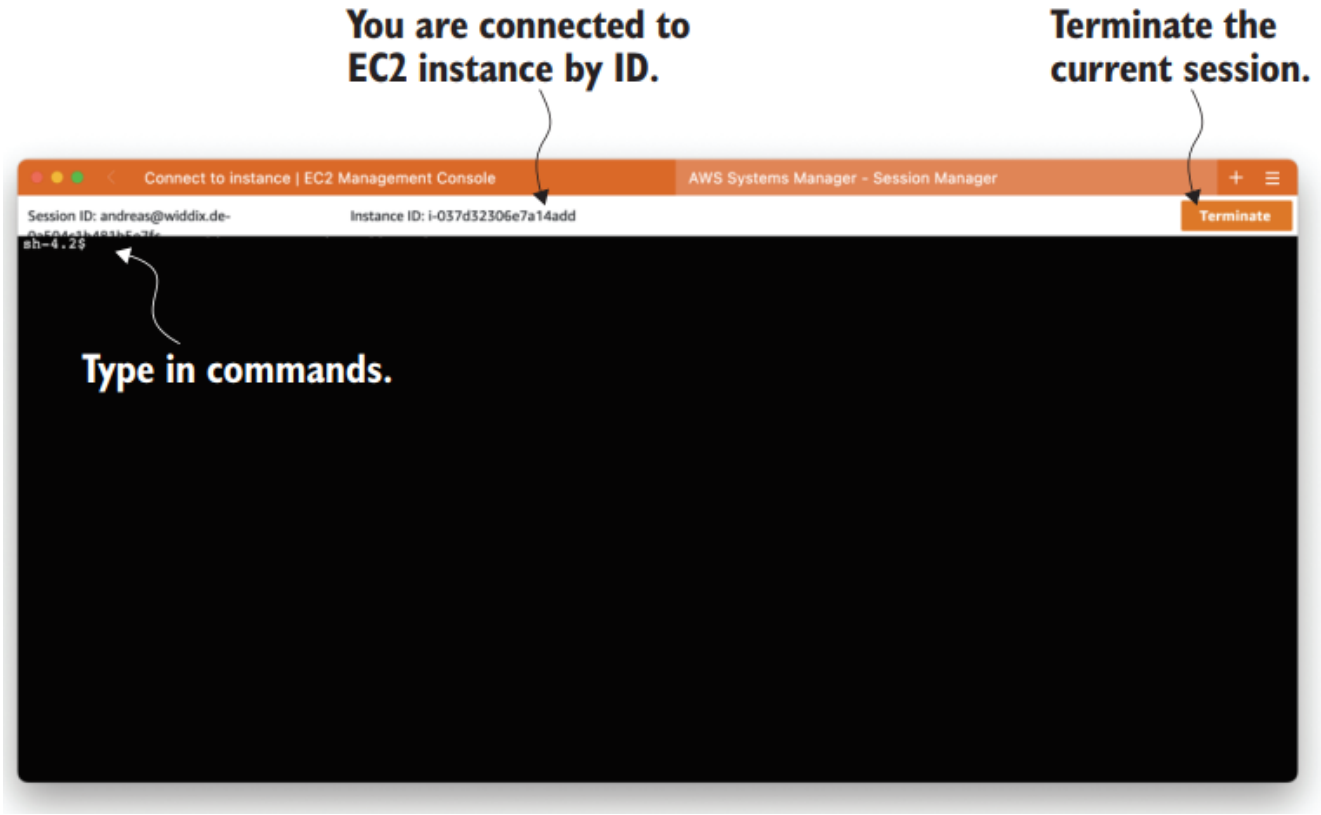
Click here to create the virtual machine.

Connect to your VM

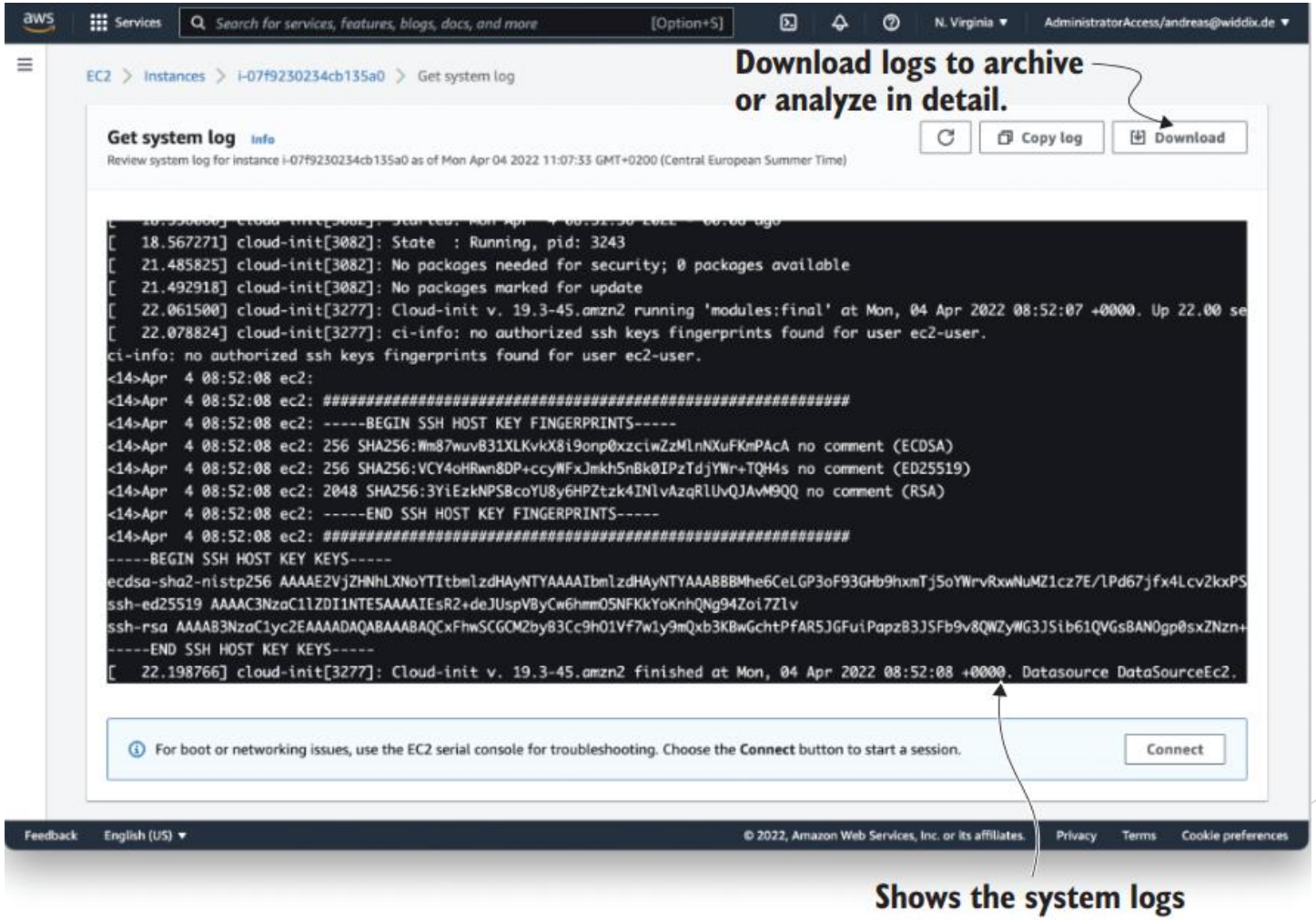




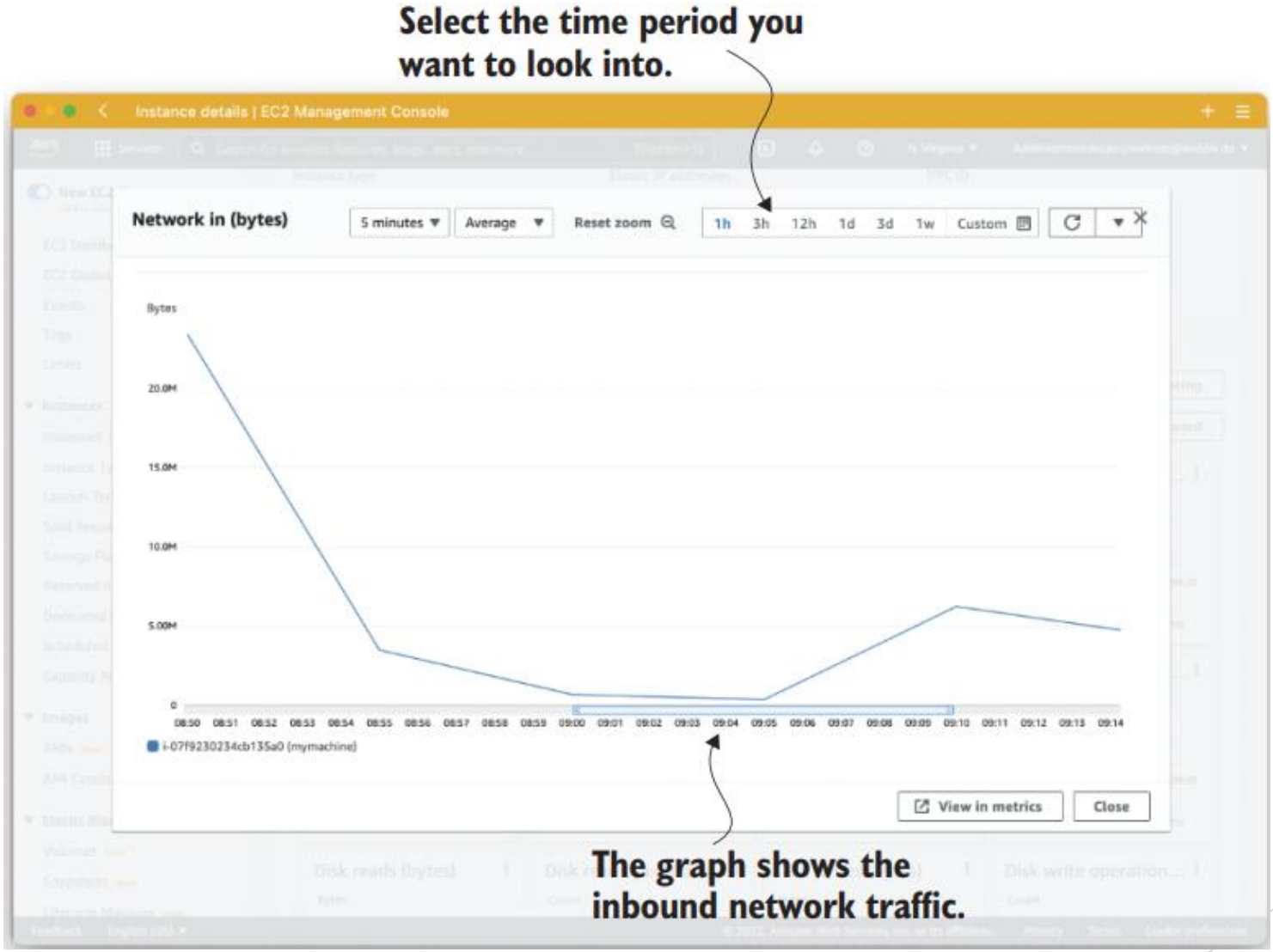
Connect to your VM



Monitoring and Debugging VM



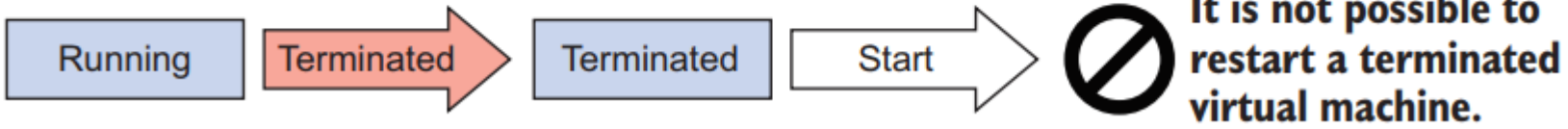
Monitoring and Debugging VM



Shutting down the VM

It is always possible to stop a running machine and to start a stopped machine.

But terminating is deleting your virtual machine.

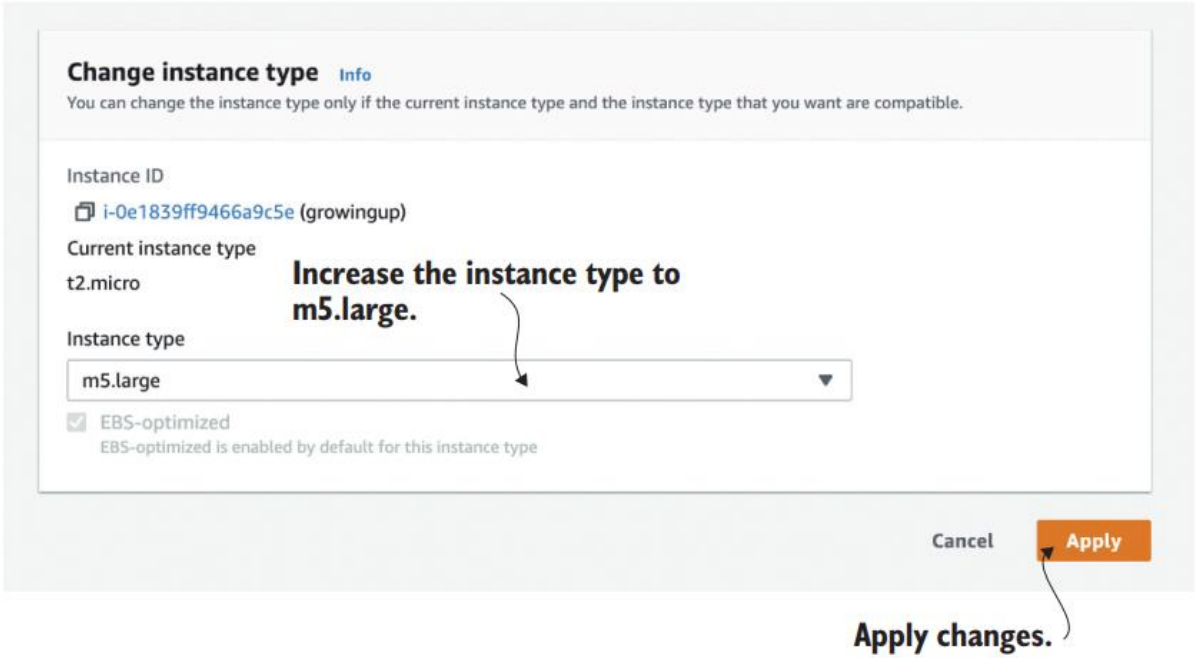


It is not possible to restart a terminated virtual machine.

Stopping or terminating unused virtual machines saves costs and prevents you from being surprised by an unexpected bill from AWS.

Change Size of VM

- After waiting for the virtual machine to stop, you can change the instance type as follows:
  1 Click the Actions button and select Instance Settings.
  2 Click Change Instance Type.
  3 Select `m5.large` as the new instance type and click the Apply button

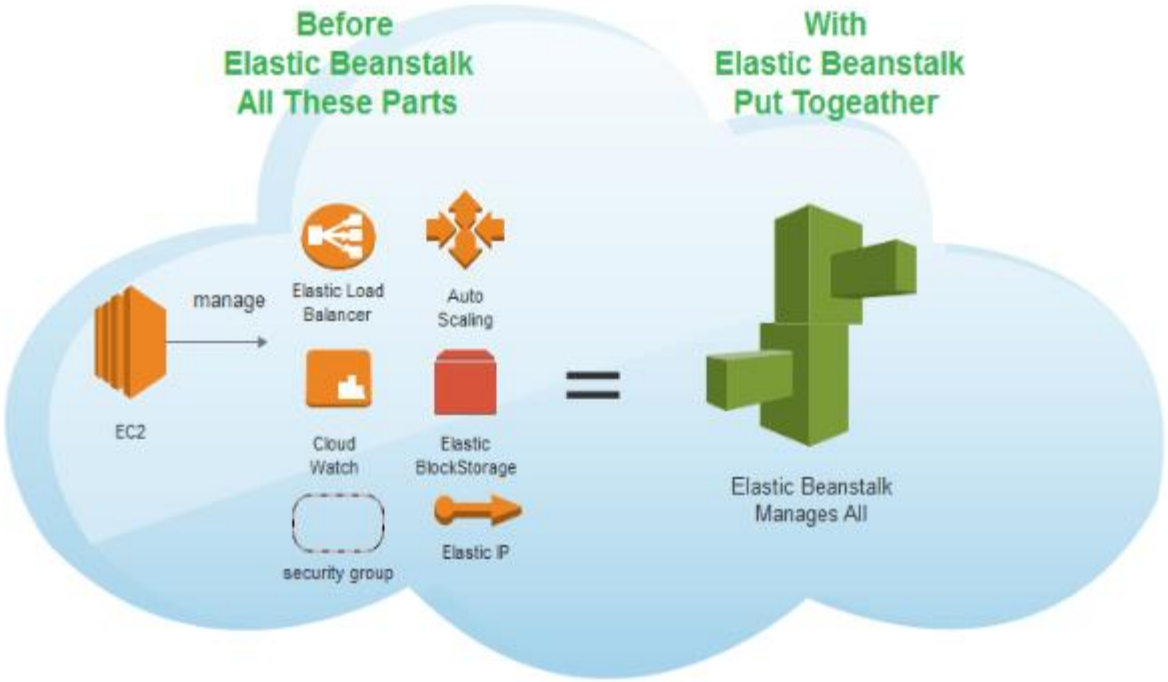


Lesson 3 Elastic Beanstalk

Michael Yang

What is Elastic Beanstalk

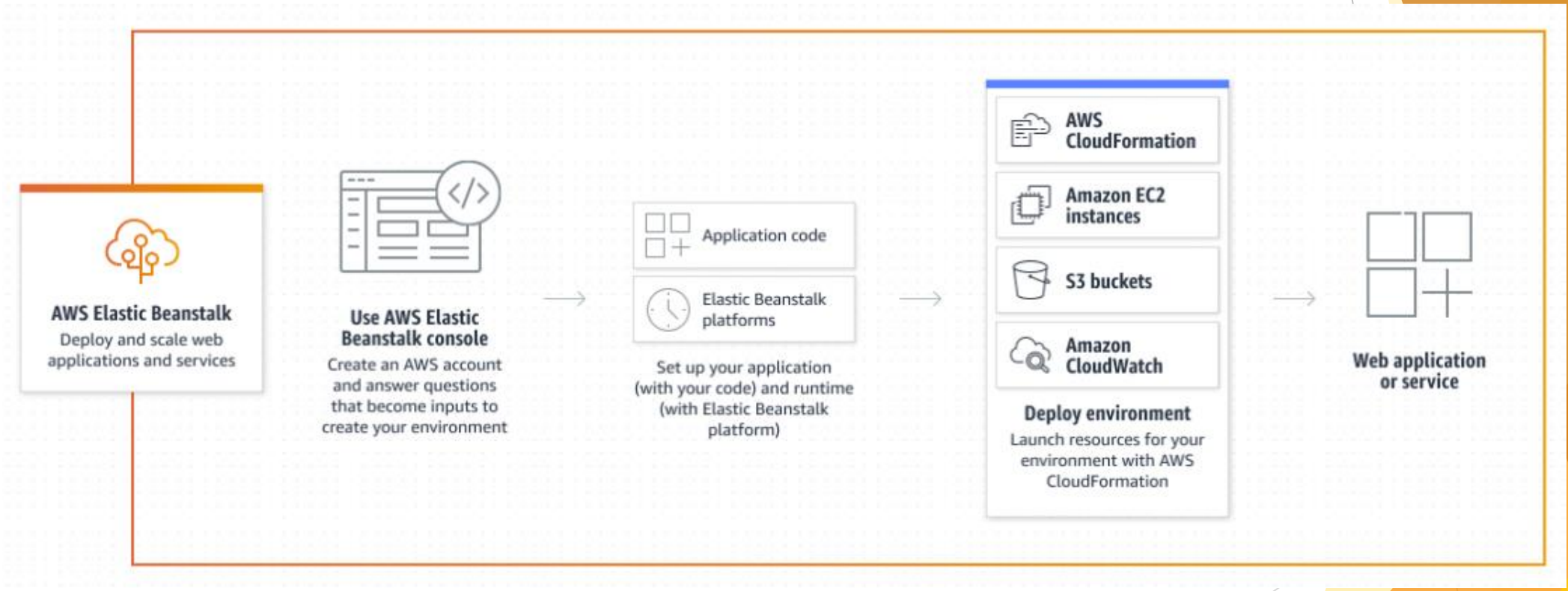
- Elastic Beanstalk is a service for deploying and scaling web applications and services. Upload your code and Elastic Beanstalk automatically handles the deployment—from capacity provisioning, load balancing, and auto scaling to application health monitoring.



Why Elastic Beanstalk

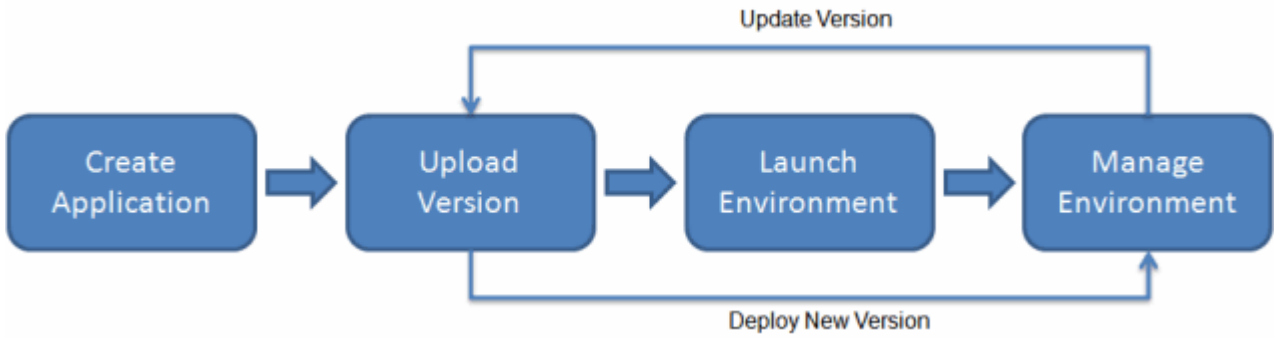
- AWS Elastic Beanstalk is the fastest way to get web applications up and running on AWS. You can simply upload your application code, and the service automatically handles details such as resource provisioning, load balancing, auto scaling, and monitoring.
- Elastic Beanstalk is ideal if you have a PHP, Java, Python, Ruby, Node.js, .NET, Go, or Docker web application. Elastic Beanstalk uses core AWS services such as Amazon Elastic Compute Cloud (EC2), Amazon Elastic Container Service (ECS), AWS Auto Scaling, and Elastic Load Balancing (ELB) to easily support applications that need to scale to serve millions of users.

How it works


AWS Elastic Beanstalk
Deploy and scale web applications and services
Use AWS Elastic Beanstalk console
Create an AWS account and answer questions that become inputs to create your environment
Application code
Elastic Beanstalk platforms
Set up your application (with your code) and runtime (with Elastic Beanstalk platform)
AWS CloudFormation
Amazon EC2 instances
S3 buckets
Amazon CloudWatch
Deploy environment
Launch resources for your environment with AWS CloudFormation
Web application or service

How to use it

- To use Elastic Beanstalk, you create an application, upload an application version in the form of an application source bundle (for example, a Java .war file) to Elastic Beanstalk, and then provide some information about the application. Elastic Beanstalk automatically launches an environment and creates and configures the AWS resources needed to run your code. After your environment is launched, you can then manage your environment and deploy new application versions. The following diagram illustrates the workflow of Elastic Beanstalk.



Components

- Application
- Application version(version)
- Environment
- Environment configuration

Application

- An Elastic Beanstalk *application* is a logical collection(container) of Elastic Beanstalk components, it contains *environments*, *versions*, and *environment configurations*. In Elastic Beanstalk an application is conceptually similar to a folder.

Application version

- In Elastic Beanstalk, an *application version* refers to a specific, labeled iteration of deployable code for a web application. An application version points to an Amazon Simple Storage Service (Amazon S3) object that contains the deployable code, such as a Java WAR file. An application version is part of an application. Applications can have many versions and each application version is unique. In a running environment, you can deploy any application version you already uploaded to the application, or you can upload and immediately deploy a new application version. You might upload multiple application versions to test differences between one version of your web application and another.

Environment

- An *environment* is a collection of AWS resources running an application version. Each environment runs only one application version at a time, however, you can run the same application version or different application versions in many environments simultaneously. When you create an environment, Elastic Beanstalk provisions the resources needed to run the application version you specified.

Environment configuration

- An *environment configuration* identifies a collection of parameters and settings that define how an environment and its associated resources behave. When you update an environment's configuration settings, Elastic Beanstalk automatically applies the changes to existing resources or deletes and deploys new resources (depending on the type of change).

Other concepts

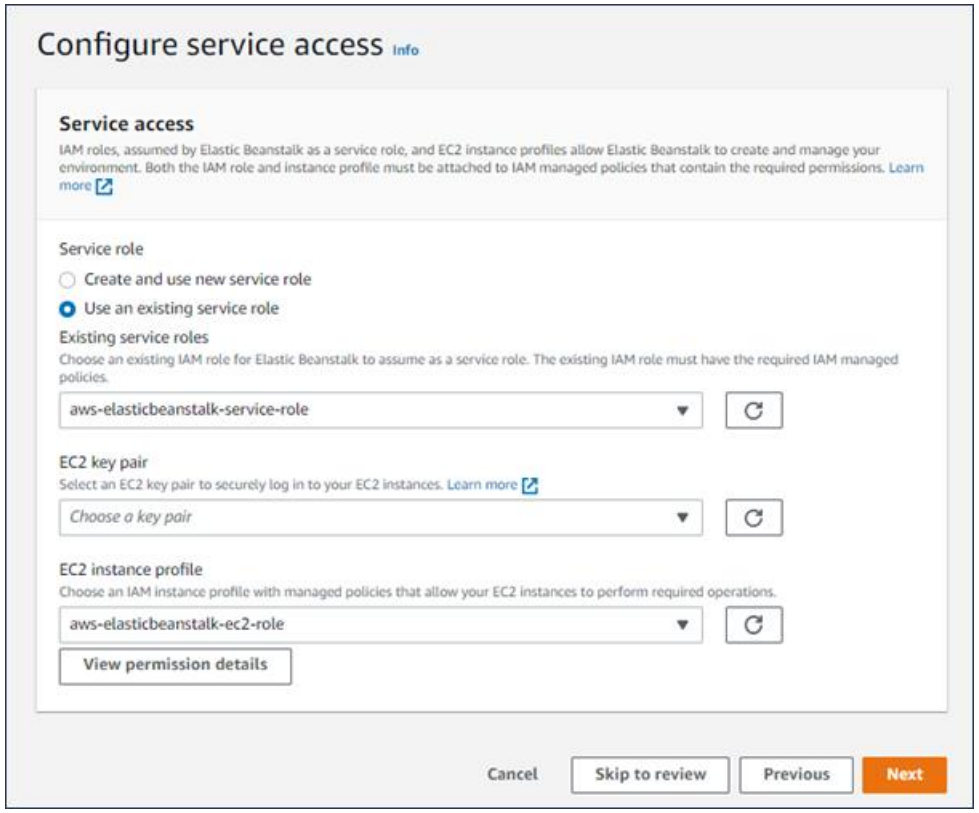
Environment tier

- When you launch an Elastic Beanstalk environment, you first choose an environment tier. The environment tier designates the type of application that the environment runs, and determines what resources Elastic Beanstalk provisions to support it. An application that serves HTTP requests runs in a web server environment tier. A backend environment that pulls tasks from an Amazon Simple Queue Service (Amazon SQS) queue runs in a worker environment tier.

Platform

- A *platform* is a combination of an operating system, programming language runtime, web server, application server, and Elastic Beanstalk components. You design and target your web application to a platform. Elastic Beanstalk provides a variety of platforms on which you can build your applications.

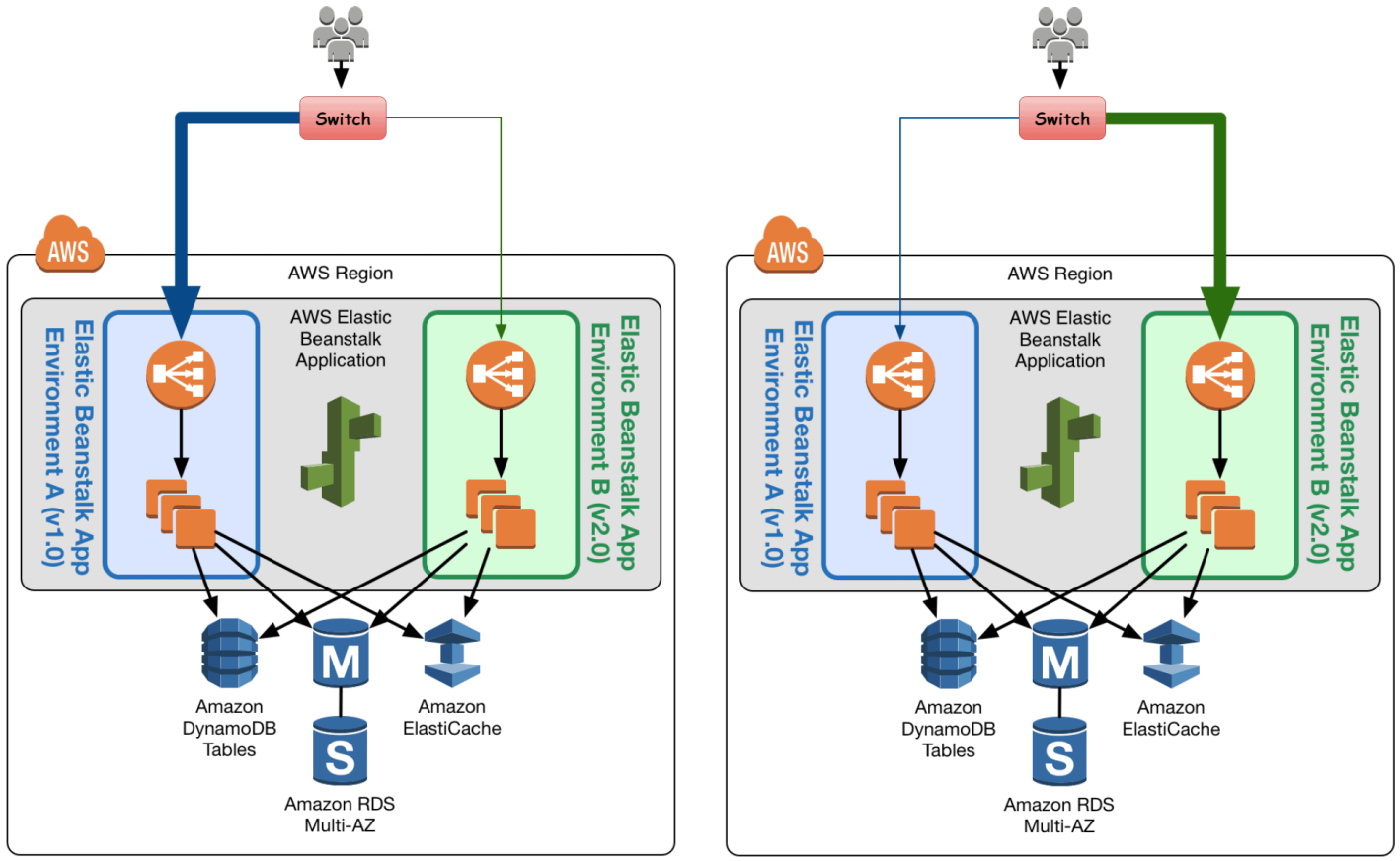
Elastic Beanstalk Security



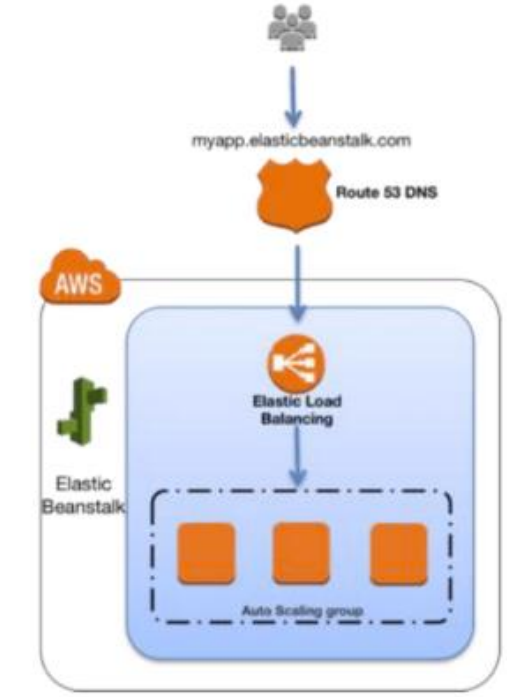
Deploy Application

- Step 1: Create an example application
- Step 2: Explore your environment
- Step 3: Deploy a new version of your application
- Step 4: Configure your environment
- Step 5: Clean up

Blue/Green deployment

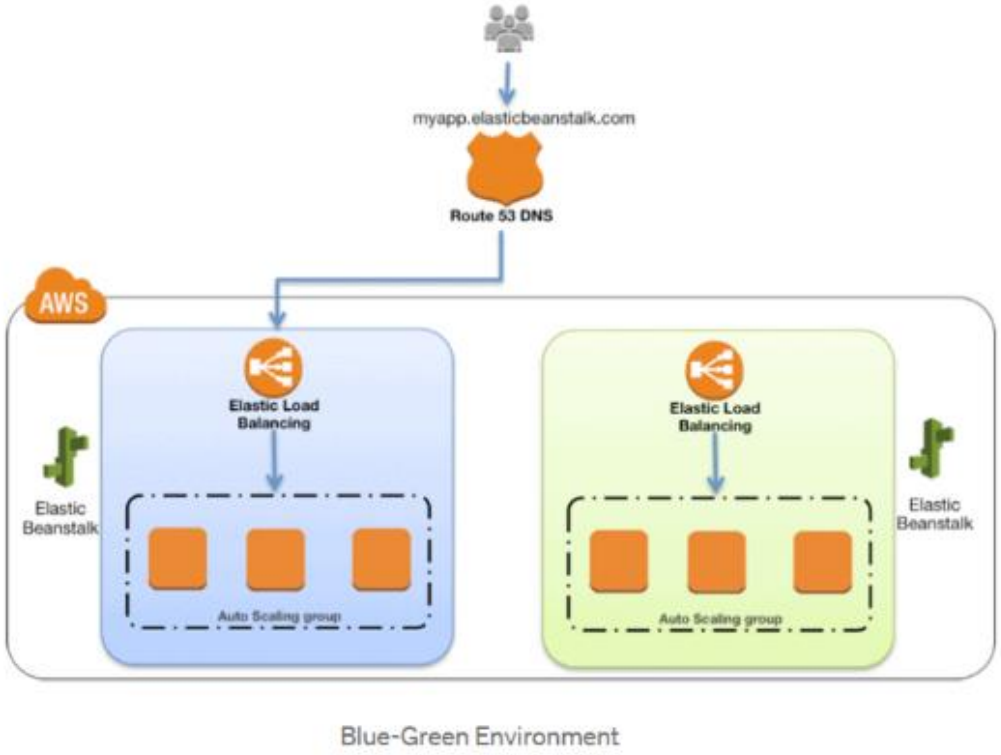


“Blue Environment”



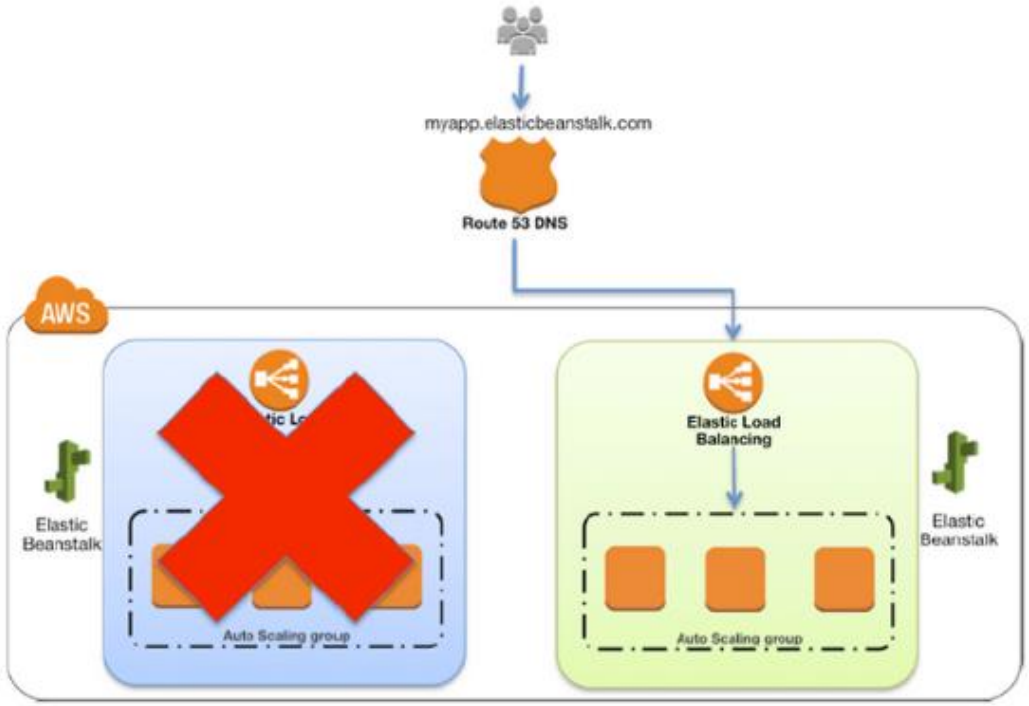
ElasticBeanStalk with Blue Environment

“Blue/Green” Environments



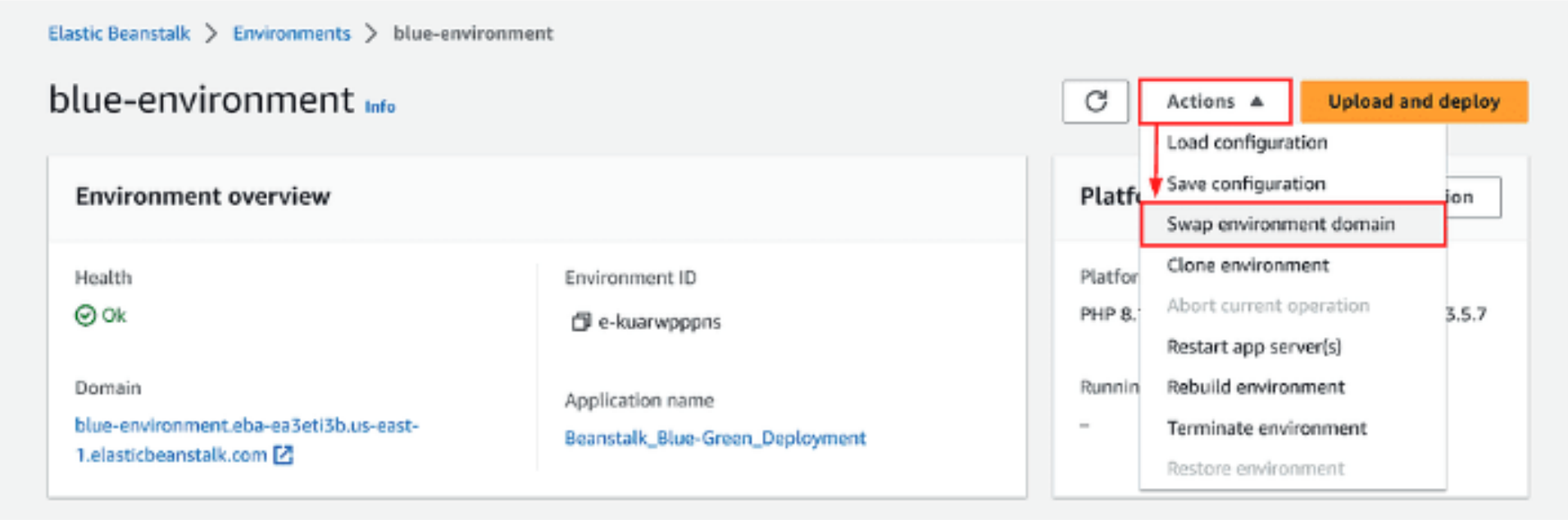
Blue-Green Environment

“Blue/Green” Environments



Live with Green Environment(Blue Terminated)

Swap the URL


Elastic Beanstalk > Environments > blue-environment
blue-environment Info
Actions
Upload and deploy
Load configuration
Save configuration
Swap environment domain
Clone environment
Abort current operation
Restart app server(s)
Rebuild environment
Terminate environment
Restore environment
Environment overview
Health
Ok
Environment ID
e-kuarwpppns
Domain
blue-environment.eba-ea3eti3b.us-east-1.elasticbeanstalk.com
Application name
Beanstalk_Blue-Green_Deployment

Lesson 4 AWS CLI and CloudFormation

Michael Yang

AWS Tools to call API

- Command-line interface (CLI)—Use the CLI to call the AWS API from your terminal.
- Software development kit (SDK)—SDKs, available for most programming languages, make it easy to call the AWS API from your programming language of choice.
- AWS CloudFormation—Templates are used to describe the state of the infrastructure. AWS CloudFormation translates these templates into API calls.

DevOps

- The DevOps movement aims to bring software development and operations together.This usually is accomplished in one of two ways:
  - Using mixed teams with members from both operations and development. Developers become responsible for operational tasks like being on call. Operators are involved from the beginning of the software development cycle, which helps make the software easier to operate.
  - Introducing a new role that closes the gap between developers and operators. This role communicates a lot with both developers and operators and cares about all topics that touch both worlds.

Automation

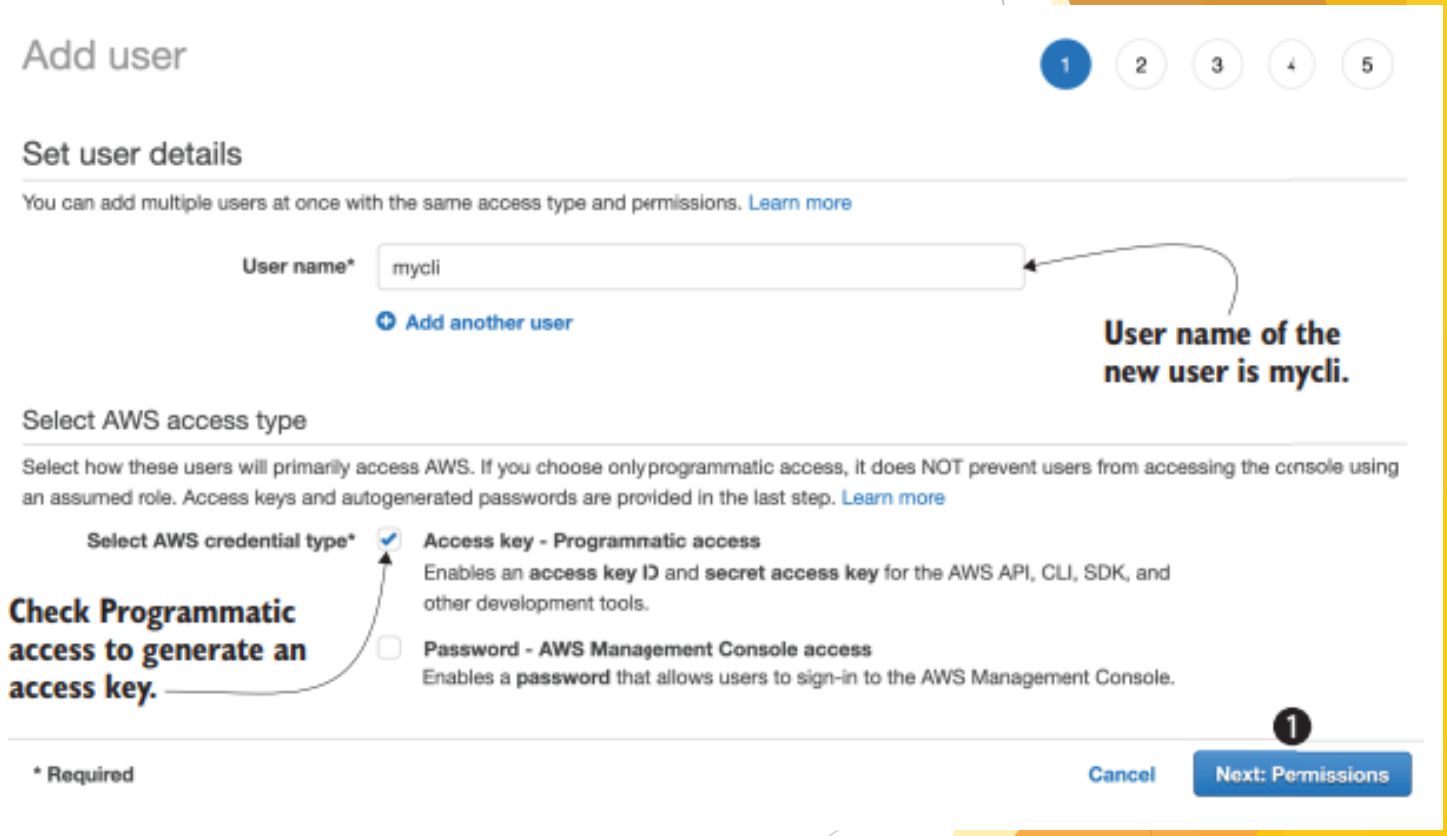
- Why should you automate instead of using the graphical AWS Management Console?
  A script or a blueprint can be reused and will save you time in the long run.
- Another benefit is that a script or blueprint is the most detailed documentation
  you can imagine (even a computer understands it).

Install AWS CLI

- The following steps guide you through installing the AWS CLI on Windows using the MSI Installer:
  1 Download the AWS CLI installer at https://awscli.amazonaws.com/AWSCLIV2.msi
  2 Run the downloaded installer, and install the CLI by going through the installation wizard.
  3 Run PowerShell as administrator by searching for “PowerShell” in the Start menu and choosing Run as Administrator from its context menu.
  4 Type `Set-ExecutionPolicy Unrestricted` into PowerShell, and press Enter to execute the command. This allows you to execute the unsigned PowerShell scripts from our examples.
- 5 Close the PowerShell window; you no longer need to work as administrator.
  6 Run PowerShell by choosing PowerShell from the Start menu.
  7 Verify whether the CLI is working by executing `aws --version` in PowerShell. The version should be at least 2.4.0.

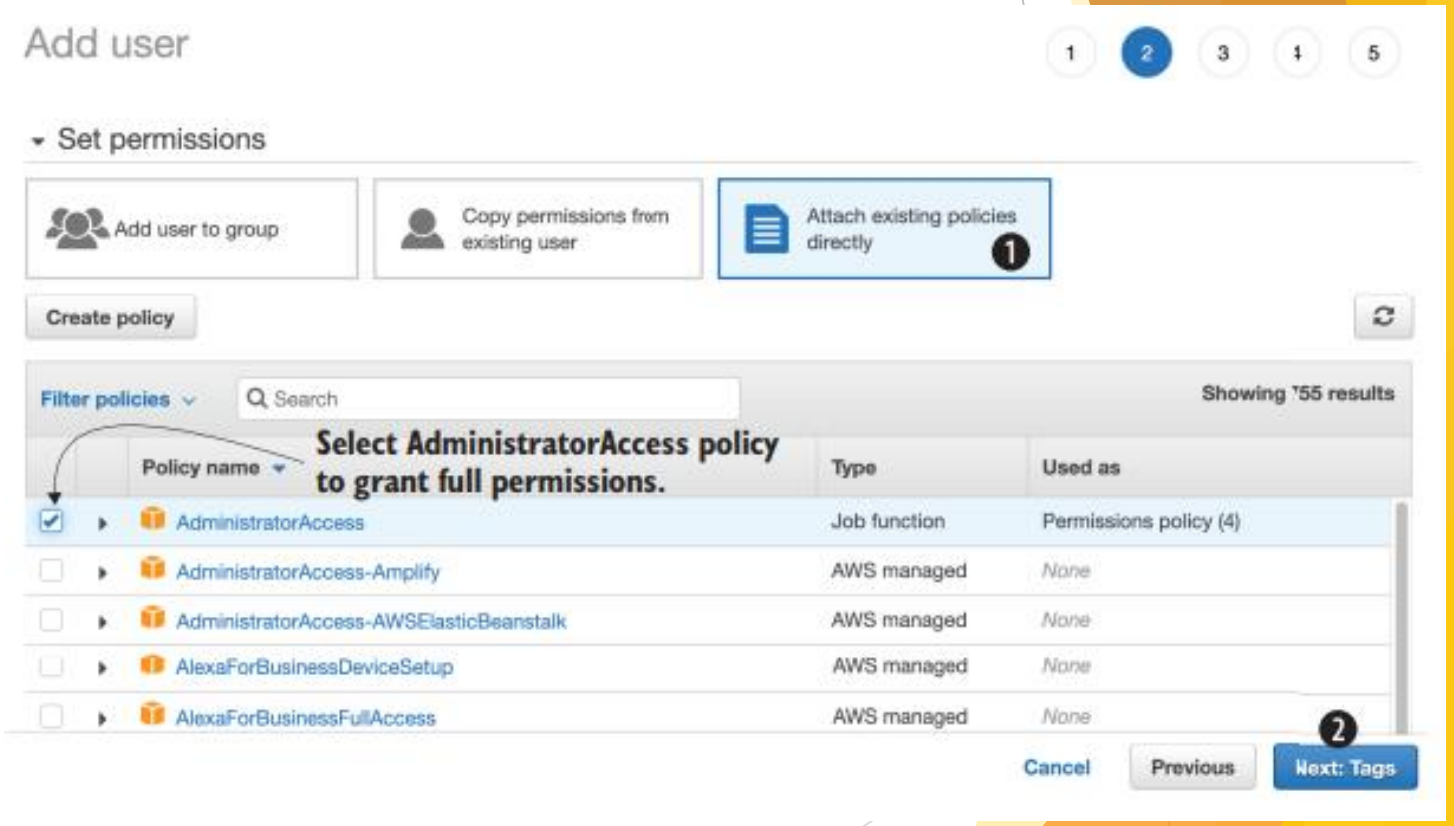
Configuring CLI

- To create a new user, use the following steps:
  1 Open the AWS Management Console at https://console.aws.amazon.com.
  2 Click Services and search for IAM.
  3 Open the IAM service.
- 1 Click Add Users to open the page shown in figure 4.4.
  2 Enter mycli as the user name.
- 3 Under AWS credential type, select Access Key—Programmatic Access.
  4 Click the Next: Permissions button.



Configuring CLI

- Define the permissions for the new user:
  1 Click Attach Existing Policies Directly.
  2 Select the AdministratorAccess policy.
  3 Click the Next: Tags button



Configuring CLI

```
$ aws configure
 AWS Access Key ID [None]:  AKIAIRUR3YLPOSVD7ZCA
 AWS Secret Access Key [None]:
➥ SSKIng7jkAKERpcT3YphX4cD87sBYgWVw2enqBj7
 Default region name [None]: us-east-1
 Default output format [None]: json
```

Your value will be different! Copy it from your browser window.

Your value will be different! Copy it from your browser window.

Infrastructure as Code

- *Infrastructure as Code* is the idea of using a high-level programming language to control infrastructures.
- Infrastructure can be any AWS resource, like a network topology, a load balancer, a DNS entry, and so on.
- In software development, tools like automated tests, code repositories, and build servers increase the quality of software engineering.
- If your infrastructure is defined as code, then you can apply these types of software development tools to your infrastructure and improve its quality.

CloudFormation Template

- A basic CloudFormation template consists of the following five parts:

Format version—The latest template format version is 2010-09-09, and this is currently the only valid value. Specify this version; the default is to use the latest version, which will cause problems if new versions are introduced in the future.

Description—What is this template about?

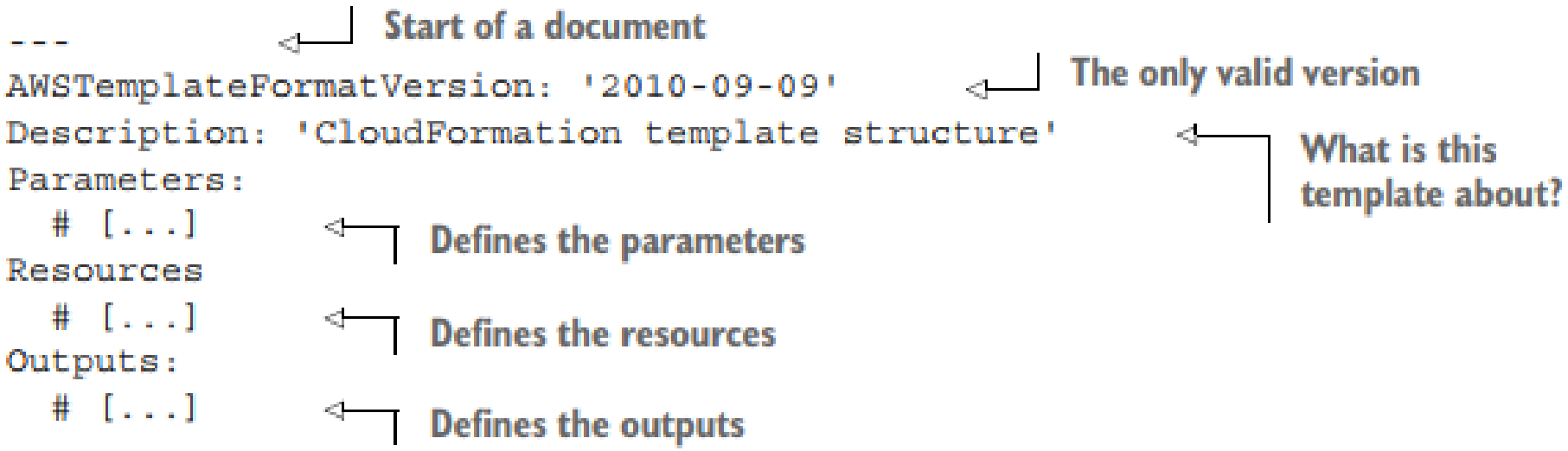
Parameters—Parameters are used to customize a template with values, for example, domain name, customer ID, and database password.

Resources—A resource is the smallest block you can describe. Examples are a virtual machine, a load balancer, or an Elastic IP address.

Outputs—An output is comparable to a parameter, but the other way around.

An output returns details about a resource created by the template, for example, the public name of an EC2 instance.

CloudFormation Template



CloudFormation Template

```
Parameters:
  Demo:
    Type: Number
    Description: 'This parameter is for demonstration'
```

You can choose the name of the parameter.

This parameter represents a number.

Description of the parameter

CloudFormation Template

- Valid types are listed

Type	Description
String CommaDelimitedList	A string or a list of strings separated by commas
Number List<Number>	An integer or float, or a list of integers or floats
AWS::EC2::AvailabilityZone::Name List<AWS::EC2::AvailabilityZone::Name>	An Availability Zone, such as us-west-2a, or a list of Availability Zones
AWS::EC2::Image::Id List<AWS::EC2::Image::Id>	An AMI ID or a list of AMIs
AWS::EC2::Instance::Id List<AWS::EC2::Instance::Id>	An EC2 instance ID or a list of EC2 instance IDs
AWS::EC2::KeyPair::KeyName	An Amazon EC2 key-pair name
AWS::EC2::SecurityGroup::Id List<AWS::EC2::SecurityGroup::Id>	A security group ID or a list of security group IDs
AWS::EC2::Subnet::Id List<AWS::EC2::Subnet::Id>	A subnet ID or a list of subnet IDs
AWS::EC2::Volume::Id List<AWS::EC2::Volume::Id>	An EBS volume ID (network attached storage) or a list of EBS volume IDs
AWS::EC2::VPC::Id List<AWS::EC2::VPC::Id>	A VPC ID (virtual private cloud) or a list of VPC IDs
AWS::Route53::HostedZone::Id List<AWS::Route53::HostedZone::Id>	A DNS zone ID or a list of DNS zone IDs

CloudFormation Template

- In addition to using the `Type` and `Description` properties, you can enhance a parameter with the properties listed in table

Property	Description	Example
`Default`	A default value for the parameter	`Default: 'm5.large'`
`NoEcho`	Hides the parameter value in all graphical tools (useful for secrets)	`NoEcho: true`
`AllowedValues`	Specifies possible values for the parameter	`AllowedValues: [1, 2, 3]`
`AllowedPattern`	More generic than `AllowedValues` because it uses a regular expression	`AllowedPattern: '[a-zA-Z0-9]*'` allows only a–z, A–Z, and 0–9 with any length
`MinLength, MaxLength`	Defines how long a parameter can be	`MinLength: 12`

CloudFormation Template

A **parameter** section of a CloudFormation template could look like this

```
Parameters:
    KeyName:
      Description: 'Key Pair name'
      Type: 'AWS::EC2::KeyPair::KeyName'
    NumberOfVirtualMachines:
      Description: 'How many virtual machine do you like?'
      Type: Number
      Default: 1
      MinValue: 1
      MaxValue: 5
    WordPressVersion:
      Description: 'Which version of WordPress do you want?'
      Type: String
      AllowedValues: ['4.1.1', '4.0.1']
```

Only key-pair names are allowed.

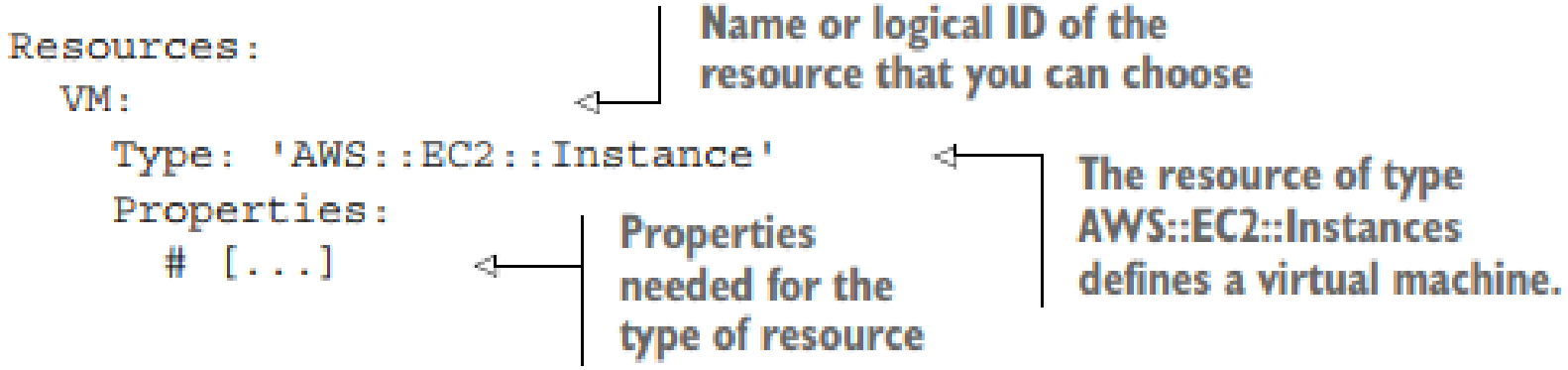
The default is one virtual machine.

Prevents massive costs with an upper bound

Restricted to certain versions

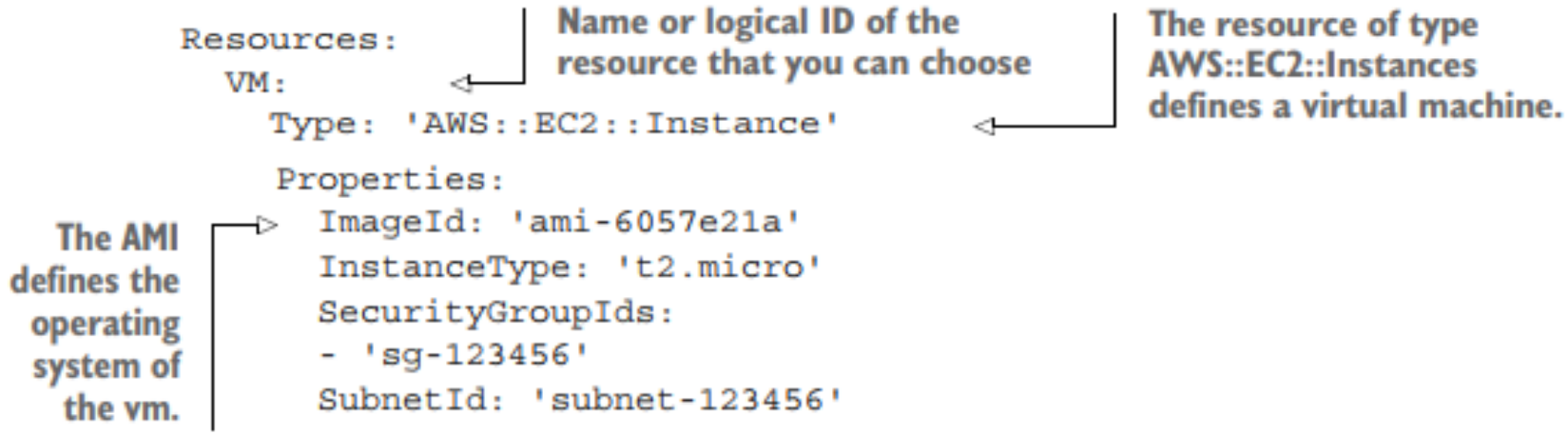
CloudFormation Template

- A **resource** has at least a name, a type, and some properties, as shown



CloudFormation Template

- When defining resources, you need to know about the type and that type's properties. In this book, you'll get to know a lot of resource types and their respective properties. An example of a single EC2 instance appears in the following code snippet



CloudFormation Template

- A CloudFormation template's **output** includes at least a name (like parameters and resources) and a value, but we encourage you to add a description as well, as illustrated in the next listing. You can use outputs to pass data from within your template to the outside.

```
Outputs:
  NameOfOutput:
    Value: '1'
    Description: 'This output is always 1'
```

Name of the output that you can choose

Value of the output

```
Outputs:
  ID:
    Value: !Ref Server
    Description: 'ID of the EC2 instance'
  PublicName:
    Value: !GetAtt 'Server.PublicDnsName'
    Description: 'Public name of the EC2 instance'
```

References the EC2 instance

Gets the attribute PublicDnsName of the EC2 instance

Lesson 5 IAM

Michael Yang

AWS Account

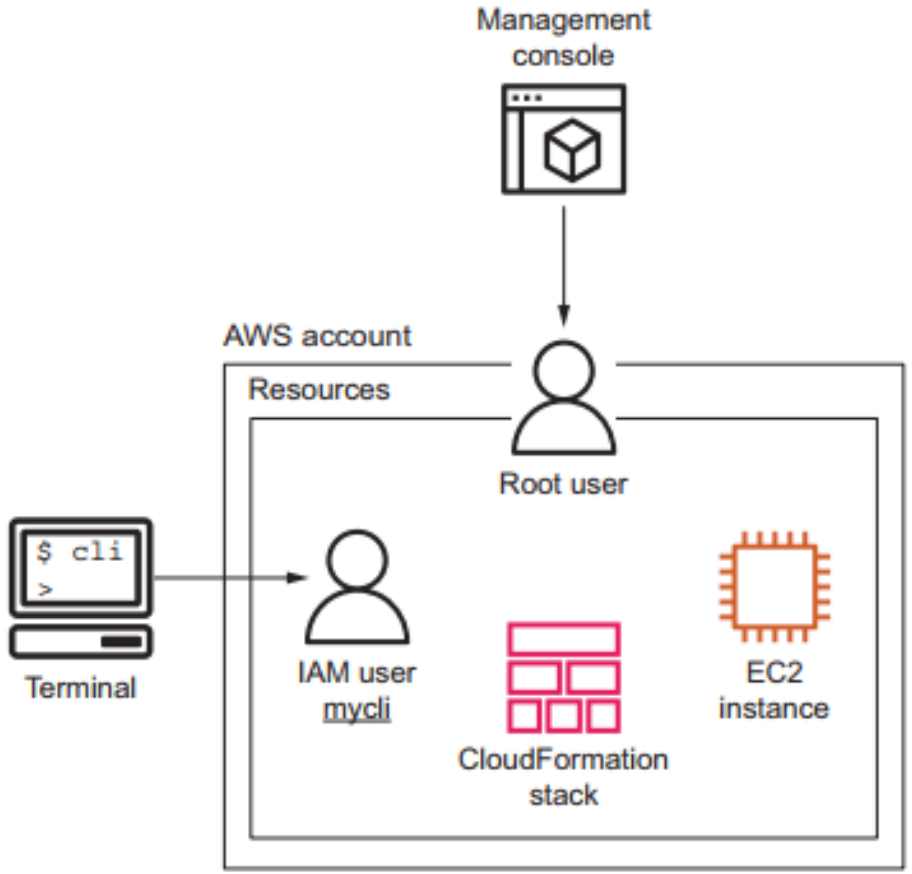
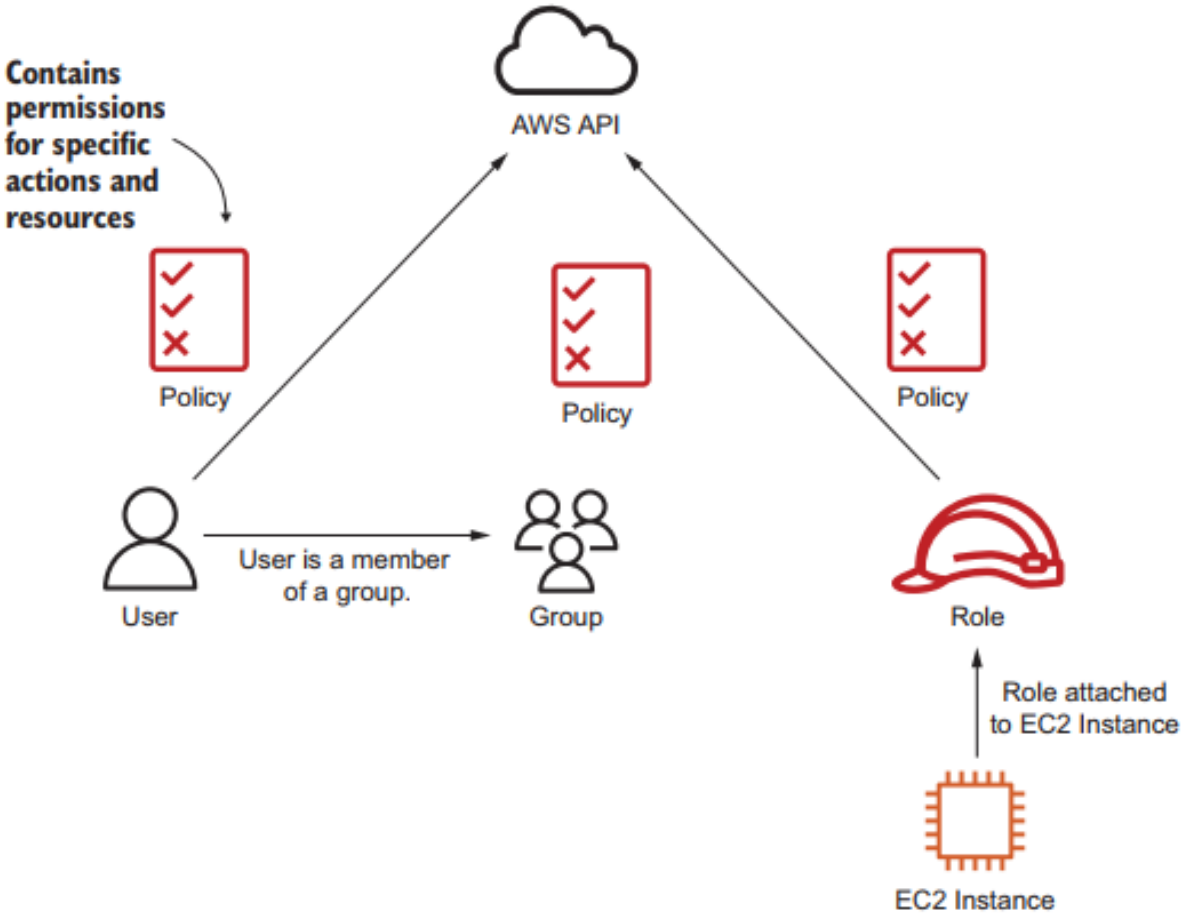
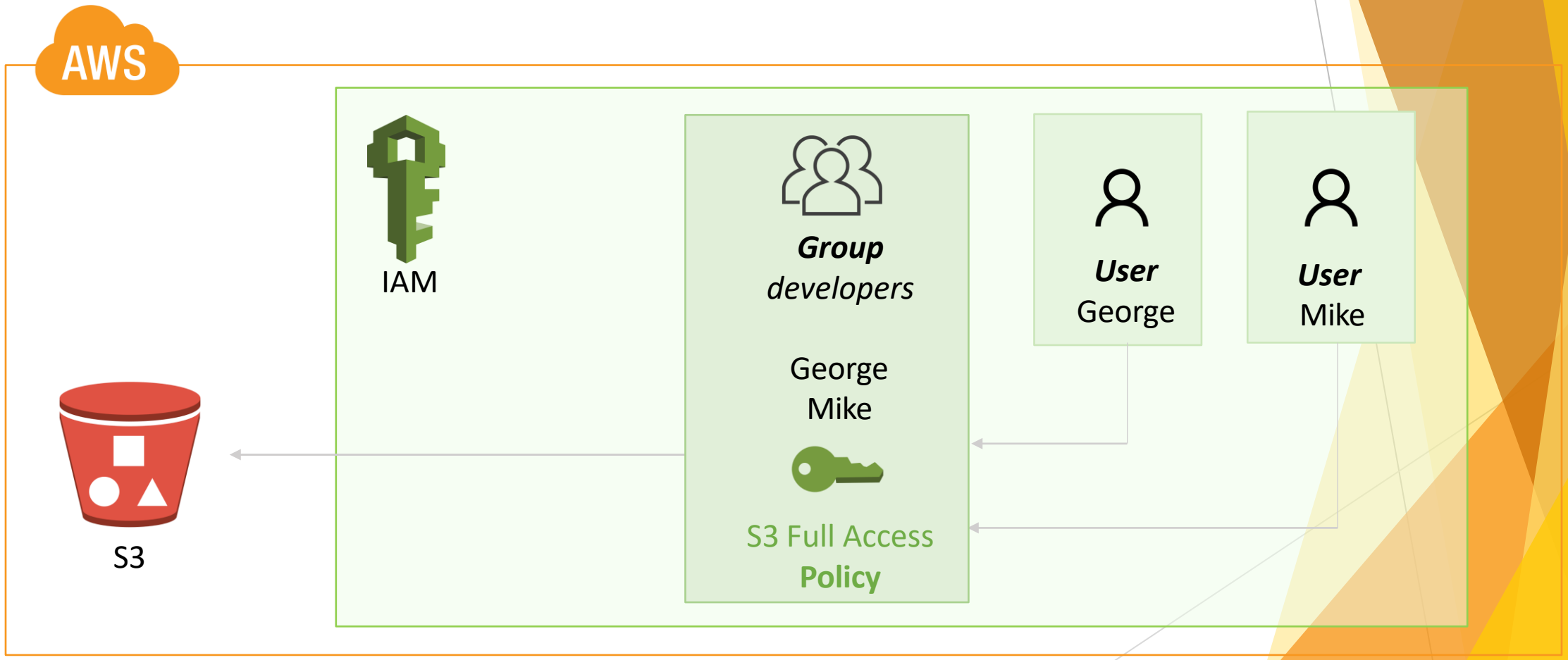


Figure 5.4 An AWS account contains all the AWS resources and comes with an AWS account root user by default.

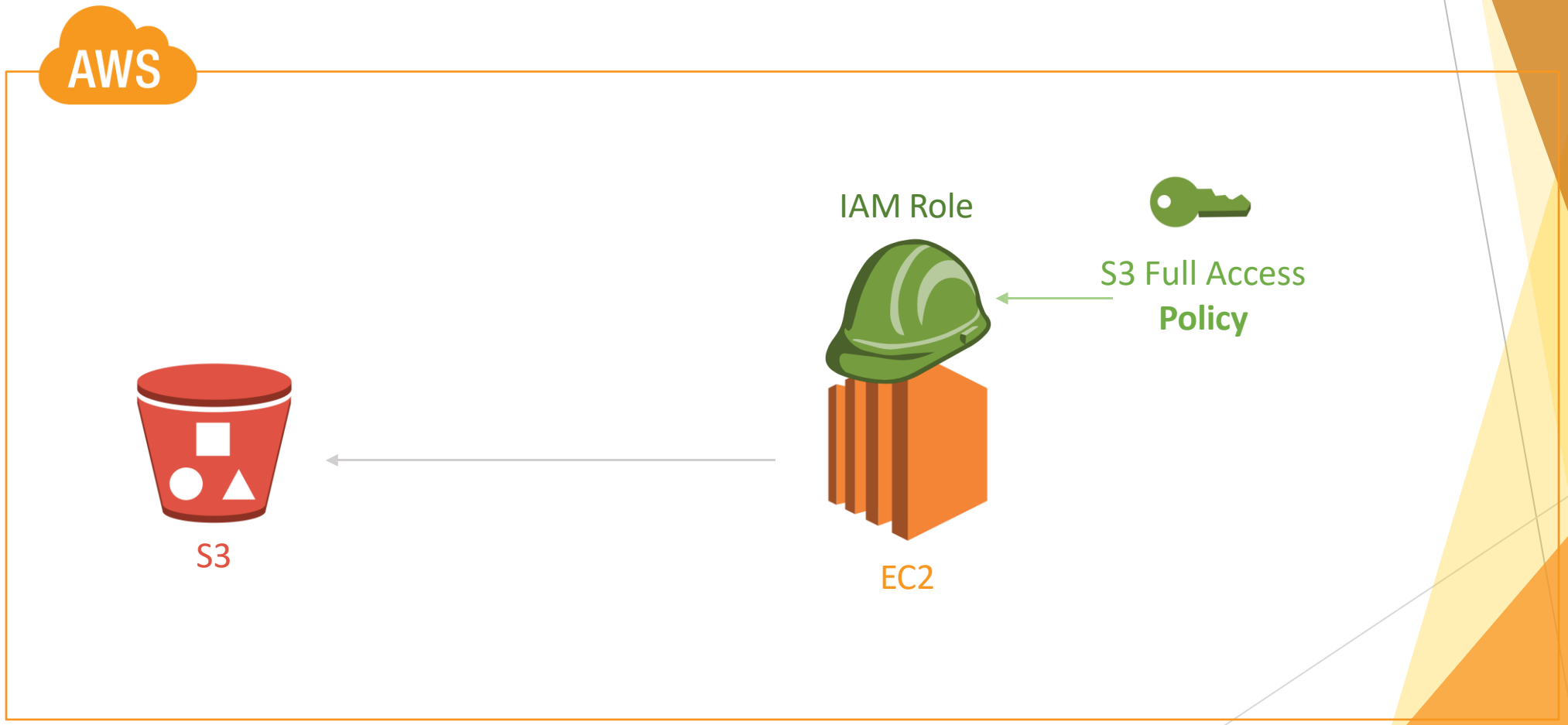
IAM Core Concepts



AWS IAM Group



AWS IAM Role



IAM user vs IAM role

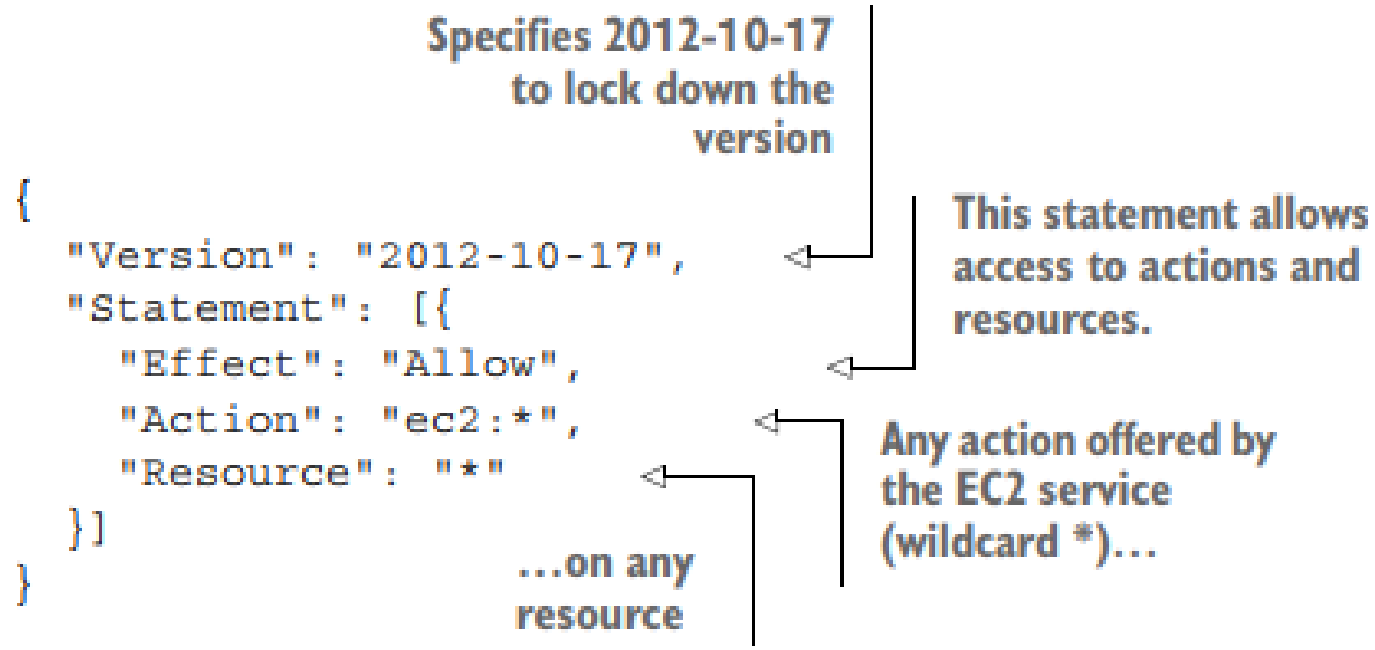
	AWS account root user	IAM user	IAM role
Can have a password (needed to log in to the AWS Management Console)	Always	Yes	No
Can have access keys (needed to send requests to the AWS API (e.g., for CLI or SDK)	Yes (not recommended)	Yes	No
Can belong to a group	No	Yes	No
Can be associated with an EC2 instance, ECS container, Lambda function	No	No	Yes

IAM Policy

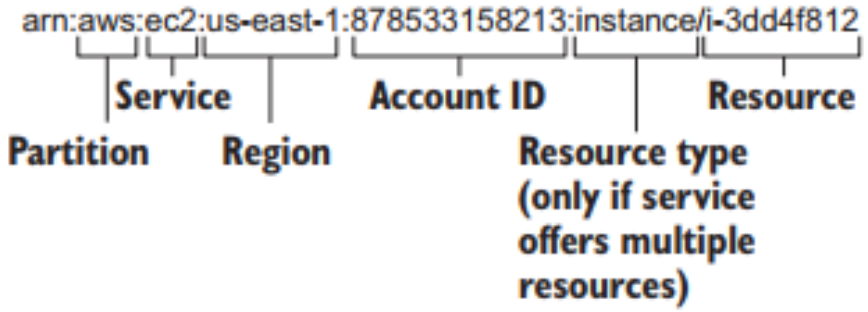
- IAM policies come in two types. *Identity policies* are attached to users, groups, or roles. *Resource policies* are attached to resources. Very few resource types support resource policies. One common example is the S3 bucket policy attached to S3 buckets.
  If a policy contains the property `Principal`, it is a resource policy. The `Principal` defines who is allowed to perform the action. Keep in mind that the principal can be set to public.

IAM Policy

- The following is an identity policy, it has one statement that allows every action for the EC2 service, for all resources:



ARN



- If you know your account ID, you can use ARNs to allow access to specific resources of a service like this:

```
{
"Version": "2012-10-17",
"Statement": [{
"Effect": "Allow",
"Action": "ec2:TerminateInstances",
"Resource": "arn:aws:ec2:us-east-
1:111111111111:instance/i-
0b5c991e026104db9"
}]
}
```

IAM Policy

- If you have multiple statements that apply to the same action, `Deny` overrides `Allow`. The following identity policy allows all EC2 actions except terminating EC2 instances

```
{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*"
    }, {
      "Effect": "Deny",
      "Action": "ec2:TerminateInstances",
      "Resource": "*"
    }]
}
```

Action is denied.

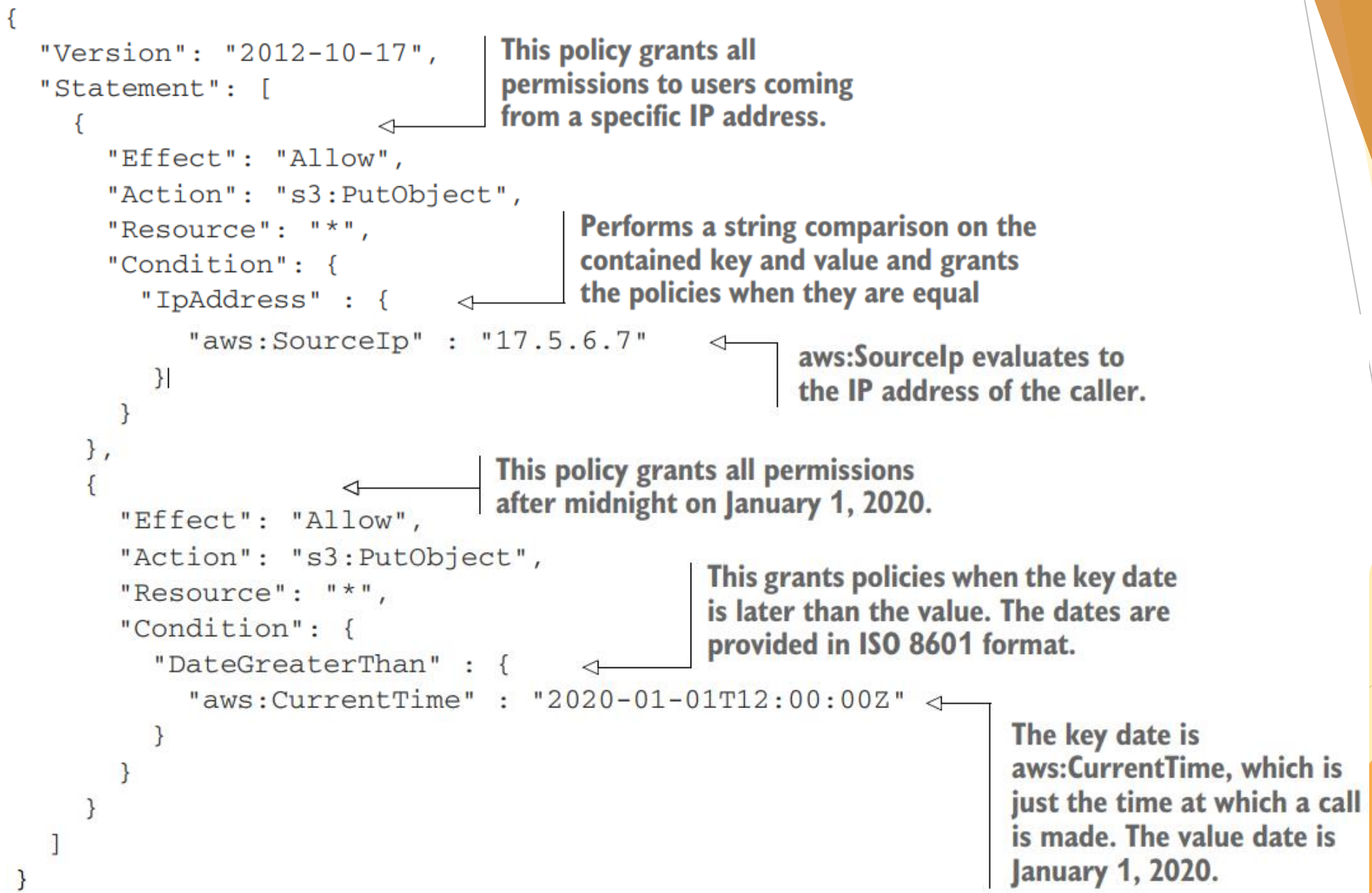
Terminate EC2 instances.

IAM policy

The Condition element (or Condition block) lets you specify conditions for when a policy is in effect.
In the Condition element, you build expressions in which you use condition operators (equal, less than, etc.) to match the condition keys and values in the policy against keys and values in the request context.
Learn more: IAM JSON policy elements: Condition

```
"Condition" : { "{condition-operator}" : { "{condition-key}" : "{condition-value}" }}
```

```
"Condition" : { "StringEqualsIgnoreCase" : { "aws:username" : "johndoe" }}
```


{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "s3:PutObject",
"Resource": "*",
"Condition": {
"IpAddress" : {
"aws:SourceIp" : "17.5.6.7"
}
}
},
{
"Effect": "Allow",
"Action": "s3:PutObject",
"Resource": "*",
"Condition": {
"DateGreaterThan" : {
"aws:CurrentTime" : "2020-01-01T12:00:00Z"
}
}
}
]
}
This policy grants all permissions to users coming from a specific IP address.
Performs a string comparison on the contained key and value and grants the policies when they are equal
aws:SourceIp evaluates to the IP address of the caller.
This policy grants all permissions after midnight on January 1, 2020.
This grants policies when the key date is later than the value. The dates are provided in ISO 8601 format.
The key date is aws:CurrentTime, which is just the time at which a call is made. The value date is January 1, 2020.

Explicit Deny wins

- The following identity policy denies all EC2 actions. The `ec2:TerminateInstances` statement isn't crucial, because `Deny` overrides `Allow`. When you deny an action, you can't allow that action with another statement:

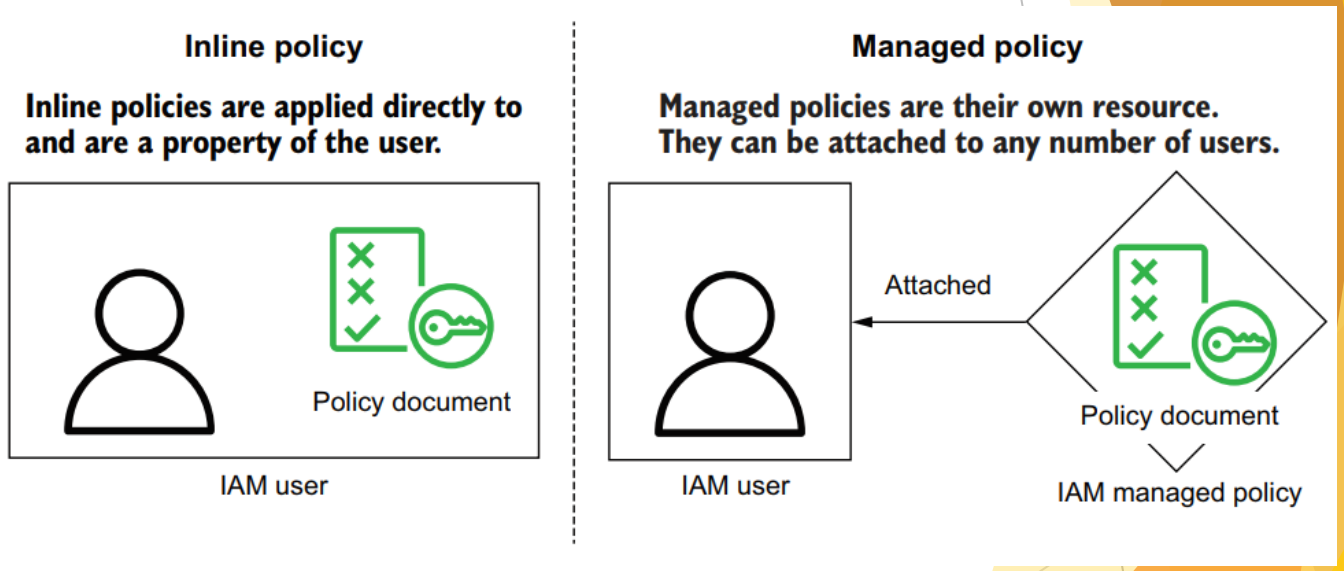
```
{
   "Version": "2012-10-17",
   "Statement": [{
     "Effect": "Deny",
     "Action": "ec2:*",
     "Resource": "*"
   }, {
     "Effect": "Allow",
     "Action": "ec2:TerminateInstances",
     "Resource": "*"
   }]
}
```

Denies every EC2 action

Allow isn't crucial; Deny overrides Allow.

Two Identity Polic

- The following two types of identity policies exist:
Managed policy—If you want to create identity policies that can be reused in your account, a managed policy is what you're looking for. There are two types of managed policies:
AWS managed policy—An identity policy maintained by AWS. There are identity policies that grant admin rights, read-only rights, and so on.
Customer managed—An identity policy maintained by you. It could be an identity policy that represents the roles in your organization, for example.
Inline policy—An identity policy that belongs to a certain IAM role, user, or group. An inline identity policy can't exist without the IAM role, user, or group that it belongs to.



Managed Policy

```
$ aws iam create-policy \
   --policy-name SamplePolicy \
   --policy-document file://policy.json

$ aws iam attach-user-policy \
   --user-name Alice
   --policy-arn arn:aws:iam::123456789012:policy/SamplePolicy
```

Creates a new managed policy

Attaches a managed policy to a user

This is the ARN of the managed policy we just created. It will be in the response of the create-policy call.

Inline Policy

```
$ aws iam create-user \
    --user-name Bob

$ aws iam put-user-policy \
    --user-name Bob \
    --policy-name SampleInlinePolicy \
    --policy-document file://policy.json
```

Puts the policy on the user Bob, created before

Names the policy for later reference

This is a local file that contains the policy document you want to use.

Resource Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:PutObject",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/Alice"
      },
      "Resource": "arn:aws:s3:::my-sample-bucket/*"
    }
  ]
}
```

This is the Principal block of the resource policy. It determines which entities the policy refers to.

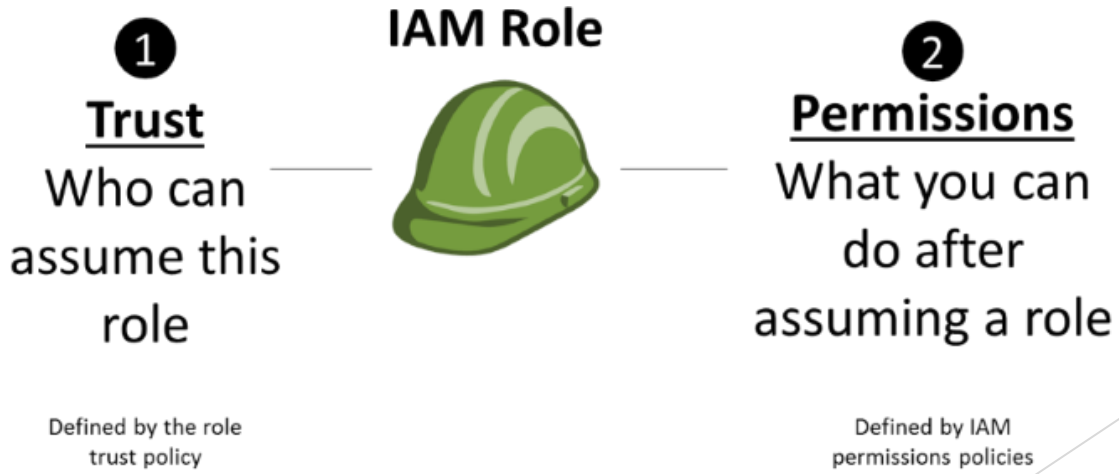
The "AWS": ARN notation is used to reference an IAM entity—in this case, a user.

The Resource block is required in bucket policies and, in this instance, refers to all objects in this bucket.

IAM role Under the hood

IAM role is similar to IAM user but you assign that to a resource, not a developer. There is a trust policy that defines which service can assume (use) that role. An IAM user and role have one thing in common, permission policies.

When the service in the trust policy assumes the role, AWS STS (Security Token Service) returns **temporary** tokens (access key id, secret access key, and **session token**), and those tokens are rotated automatically.



AWS Security Token Service (AWS STS)

Under the hood, tokens are generated and used to access AWS services. STS is a web service that enables you to request **temporary** credentials for IAM role. The example is AWS Academy, you receive a temporary token.

The temporary credentials consist of:

1. **Access key ID** - Access keys are long-term credentials for an IAM user. Like username.
2. **Secret access key** - Like password.
3. **Session token** - Validates temporary credentials.
4. **Duration** - defines how long the temporary credentials lasts. Most cases 12 hours. 15-min is min. You will not see these in the permanent tokens for users.

STS - AssumeRole

The underlying service makes an implicit “AssumeRole” call to STS on behalf of the resource and fetch the temporary tokens. It all happens under the hood.

For instance, when fetching images from S3 in EC2, EC2 makes the AssumeRole call to STS, then receives the token and provides the token to S3. You don’t have to manually rotate it. The rotation is done by AWS.

IAM User vs Role

IAM User	**IAM Role**
IAM entity assigned to a **person**.	IAM entity assigned to a **service**. It has a trust policy that specifies what services can use the role.
Tokens are permanent. It is on you to rotate that regularly.	**Tokens are temporary**. Tokens are generated by AWS STS. It has an extra token “aws_session_token”.

Exercise

```
[ec2-user@ip-172-31-17-250 ~]$ aws --version
aws-cli/1.18.147 Python/2.7.18 Linux/4.14.225-169.362.amzn2.x86_64 botocore/1.18.6
[ec2-user@ip-172-31-17-250 ~]$ aws iam list-users
Unable to locate credentials. You can configure credentials by running "aws configure".
[ec2-user@ip-172-31-17-250 ~]$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [None]: ^C
[ec2-user@ip-172-31-17-250 ~]$ aws iam list-users
{
    "Users": [
        {
            "UserName": "stephane",
            "PasswordLastUsed": "2021-04-19T20:41:59Z",
            "CreateDate": "2021-04-19T20:38:18Z",
            "UserId": "AIDATCOXNUAWA5KFV6MM7",
            "Path": "/",
            "Arn": "arn:aws:iam::211442049068:user/stephane"
        }
    ]
}
[ec2-user@ip-172-31-17-250 ~]$
```

IAM Summary

IAM is about:

1. allow/deny
2. what actions?
3. on which resources?
4. who?
    a. principle propriety (resource-based)
    b. IAM user or role (identity-based)
5. condition (optional)

Lesson 6 VPC

Michael Yang

What is VPC

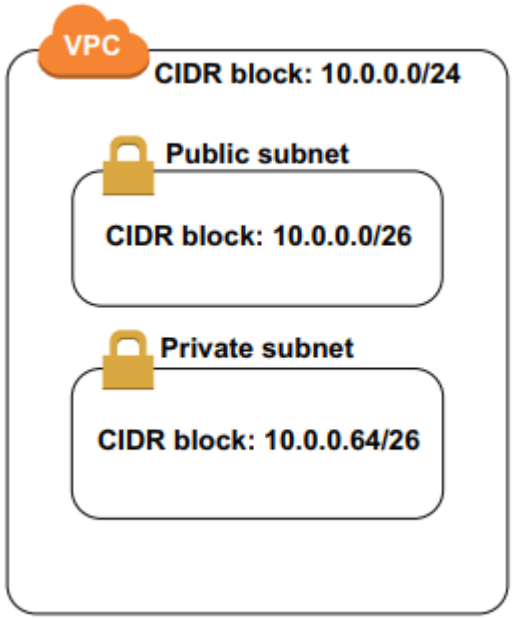
- A VPC is a virtual network. If you want to create any networked resources in AWS, you're going to first have to create a VPC.
- This is a relatively easy task, as a VPC has only a couple of options. The main one is the CIDR block. This is the range of IP addresses that will be available for use in your network. CIDR stands for classless inter-domain routing

CIDR block	Equivalent IP range	Note
10.0.0.0/24	10.0.0.0–10.0.0.255	Contains the 256 addresses, starting at 10.0.0.0
192.168.1.1/32	192.168.1.1	Only contains 1 address
0.0.0.0/0	0.0.0.0–255.255.255.255	Covers the entire IPv4 address space

Create a VPC

```
$ aws ec2 create-vpc \
--cidr-block 10.0.0.0/24
```

Create a Subnet



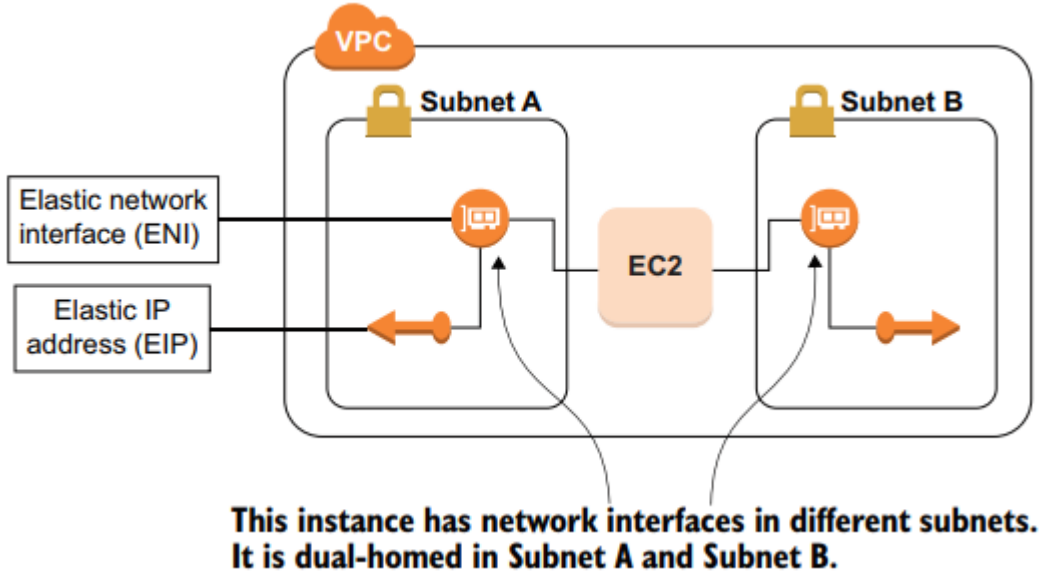
This is the ID of the VPC created earlier.

```
$ aws ec2 create-subnet \
    --vpc-id vpc-1234
    --cidr-block 10.0.0.0/26

$ aws ec2 create-subnet \
    --vpc-id vpc-1234
    --cidr-block 10.0.0.64/26
```

NIP

- Elastic network interfaces are the virtual equivalent of an NIC or network card on a physical machine. These ENIs are the connection between networked resources like EC2 instances and your virtual network. In fact, you can attach additional ENIs to your instances, and those ENIs can be in two different subnets. This creates what are called *dual-homed* instances, which can be visualized below



ENI

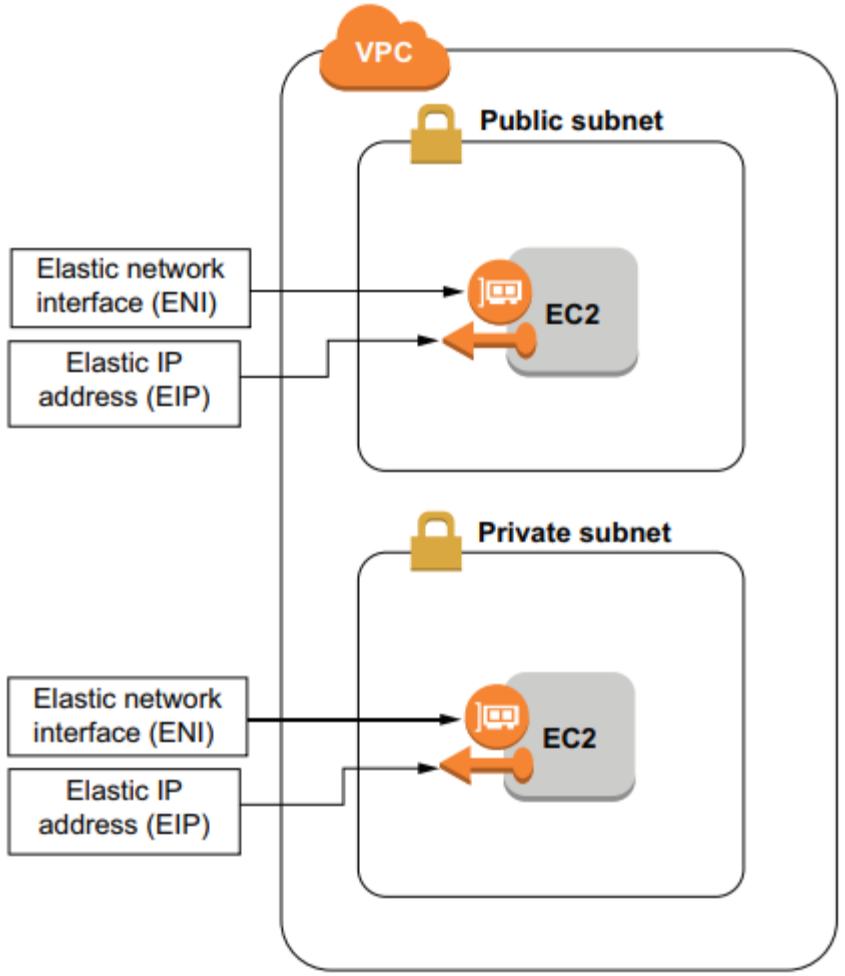
```
$ aws ec2 run-instances \
--instance-type t2.micro \
--subnet-id subnet-1234
--image-id ami-1234

$ aws ec2 run-instances \
--instance-type t2.micro \
--subnet-id subnet-5678
--image-id ami-1234 #B
```

Replace with the ID of the public subnet.

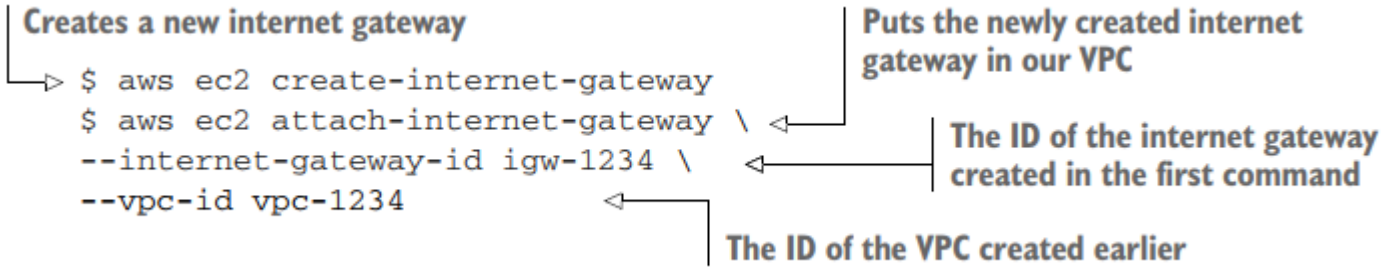
Replace with the ID of the AMI you want to use.

Replace with the ID of the private subnet.



Internet gateway

- An internet gateway, sometimes called an IGW, is a resource that is created in a VPC. When an internet gateway is attached to a VPC, then traffic can be routed from inside the VPC to the internet through that IGW, and vice versa.



Internet gateway

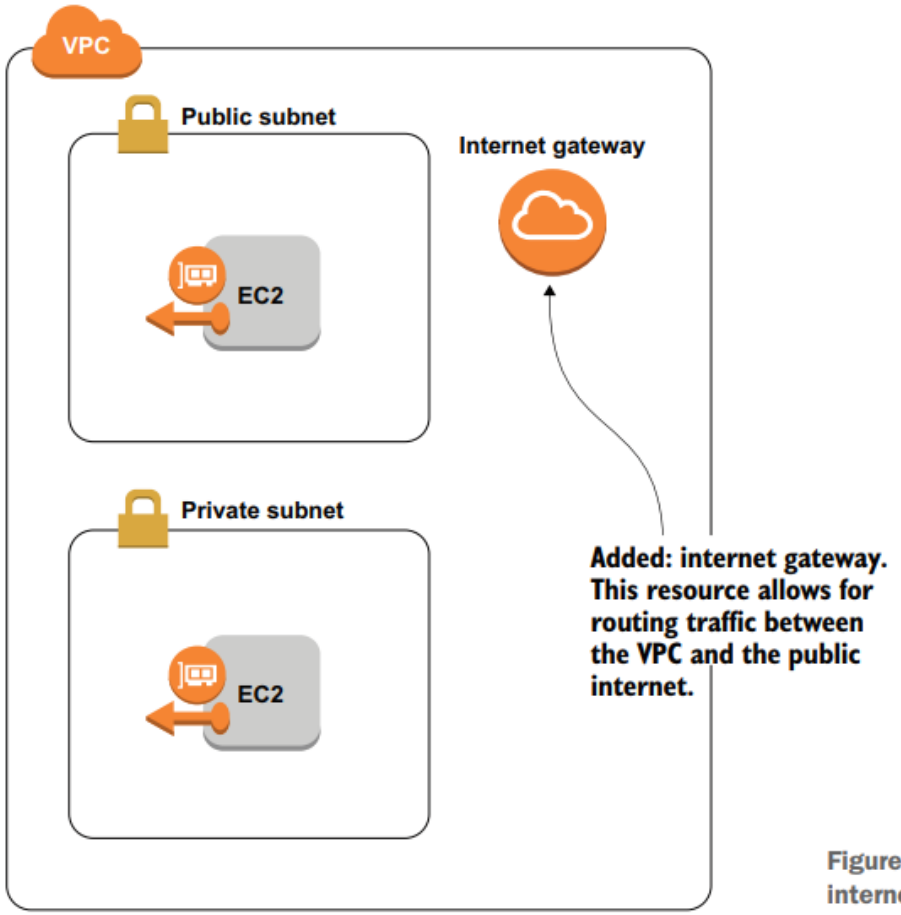
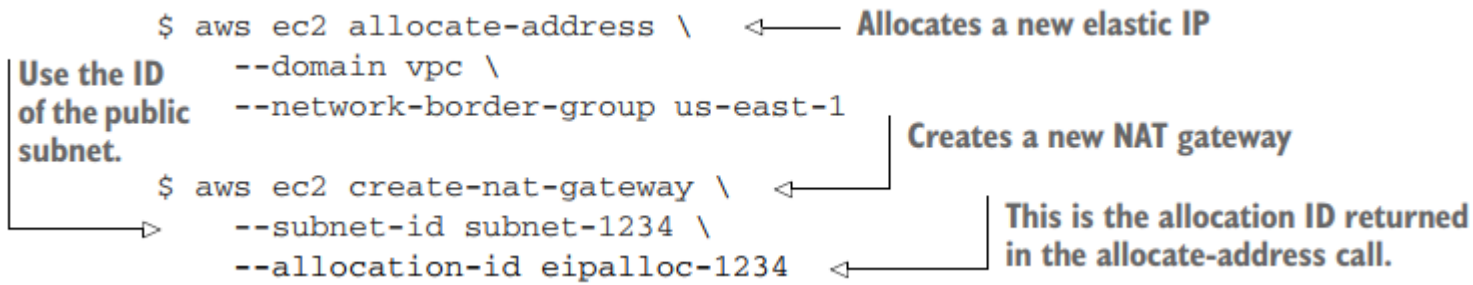


Figure 5.5 Adding an internet gateway to a VPC

NAT gateway

- A NAT gateway is not an alternative to an internet gateway; they actually work in tandem.



NAT gateway

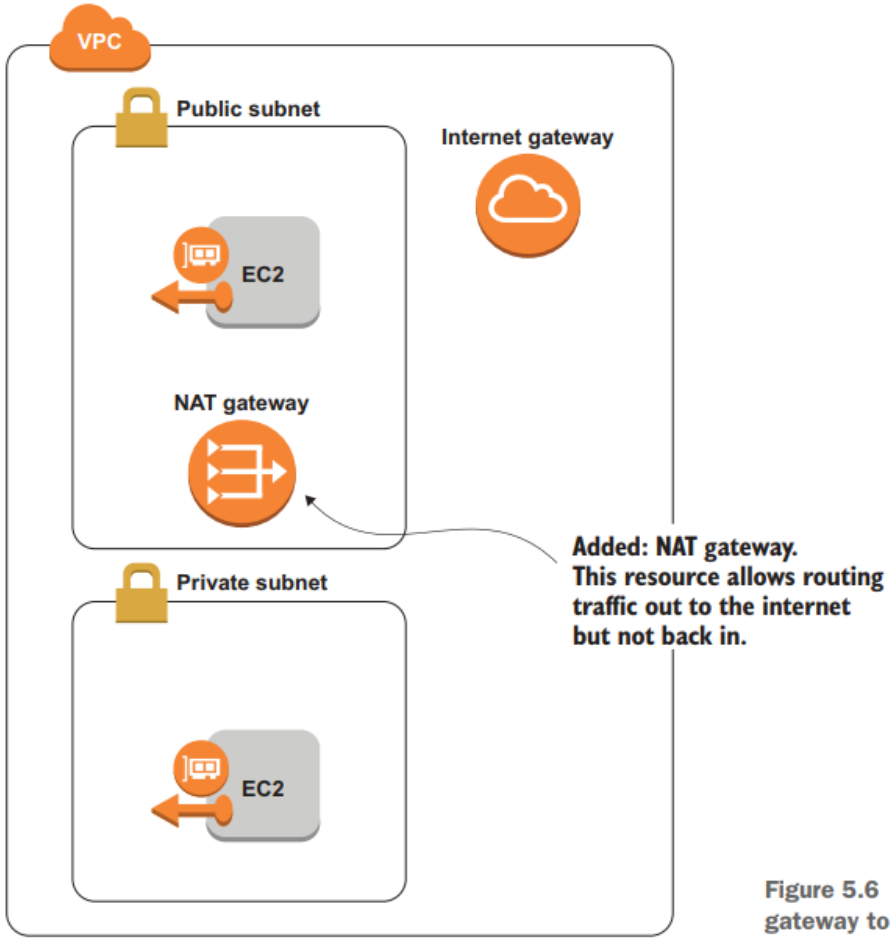


Figure 5.6 Adding a NAT gateway to a public subnet

Route table

- A route table defines how traffic is routed throughout your VPC. It is a set of rules that say where traffic should be directed based on the IP address it was sent to. Each of these rules is aptly called a *route*. Every route consists of two parts: a destination and a target.

- Whenever you create a VPC, a route table is automatically created and attached to that VPC for you.

Route	Destination	Target
Route 1	10.0.0.0/26	local
Route 2	0.0.0.0/0	igw-1234

Route tables

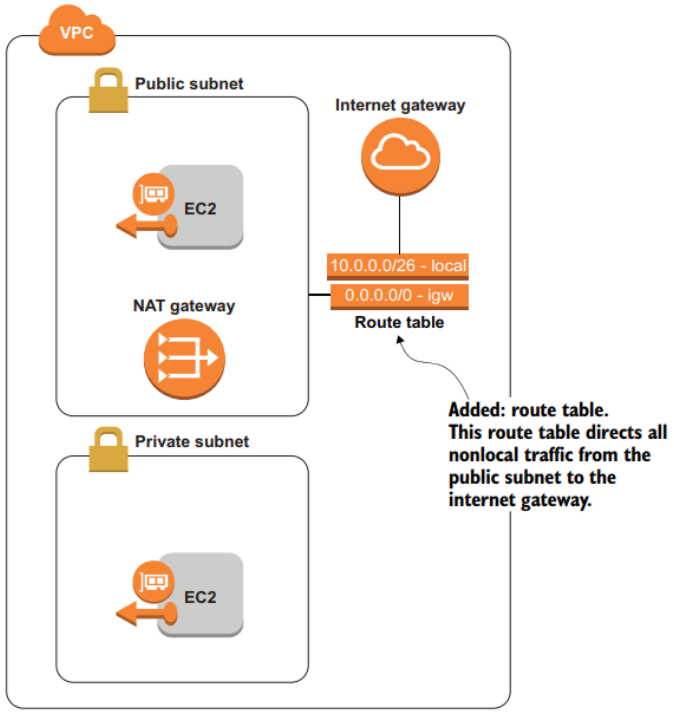
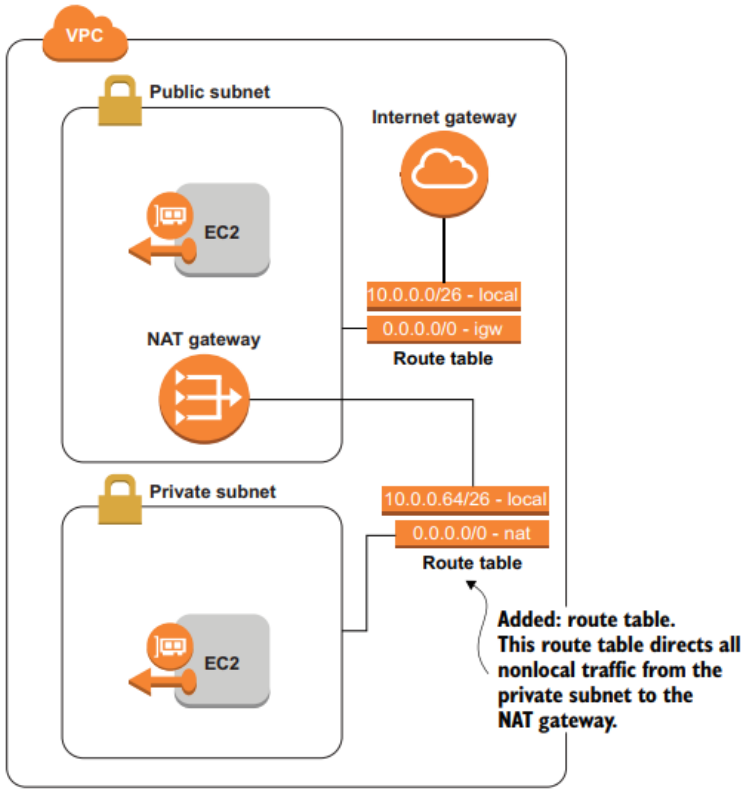
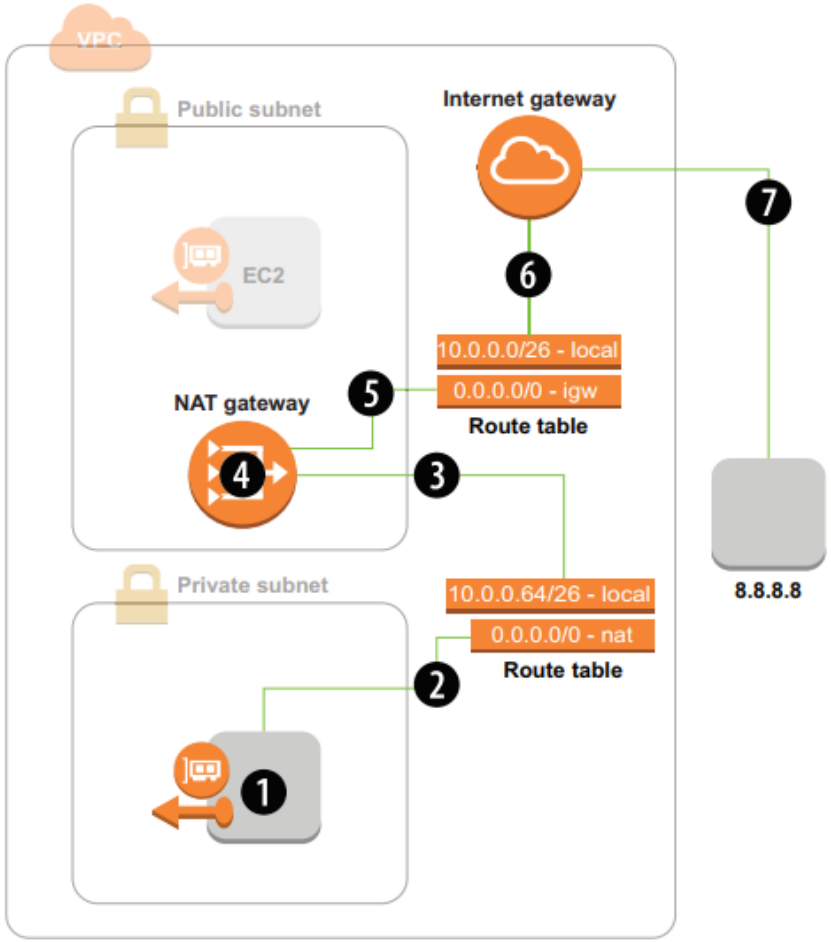


Figure 5.8 Using a route table to connect a subnet to an internet gateway. The route table has a default route that directs local traffic within the VPC, and a custom route that directs all other traffic through the IGW.

Route table


VPC
Public subnet
Internet gateway
EC2
10.0.0.0/26 - local
0.0.0.0/0 - igw
Route table
NAT gateway
Private subnet
10.0.0.64/26 - local
0.0.0.0/0 - nat
Route table
EC2
Added: route table.
This route table directs all
nonlocal traffic from the
private subnet to the
NAT gateway.

- The path of traffic as it flows between an instance in a private subnet to a server in the public internet, leveraging NAT and internet gateways



Security Group

- A *security group* is a set of rules that determine what network traffic is allowed in and out of an instance.
- There are two kinds of rules in a security group, inbound and outbound, and each is made up of three key elements.
- An example TCP outbound rule might look like what is shown in table

Destination	Protocol (number)	Port range
0.0.0.0/0	TCP (6)	443

Security Group

Inbound security group rule allowing TCP traffic on port 22 from anywhere

Source	Protocol (number)	Port range
0.0.0.0/0	TCP (6)	22

When you create a VPC, a default security group is created.

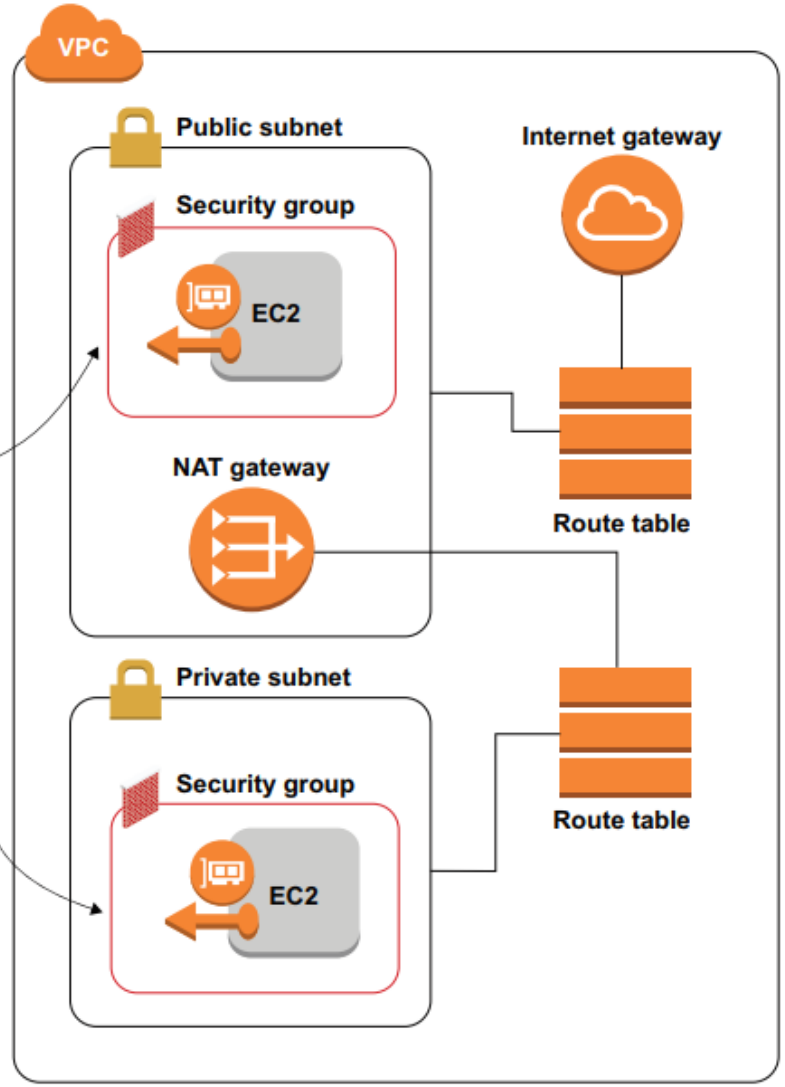
Type	Source/destination	Protocol	Port range
Inbound	self	All	All (0-65535)
Outbound	0.0.0.0/0	All	All (0-65535)

Bad Inbound

Source	Protocol (number)	Port range
0.0.0.0/0	TCP (6)	80
0.0.0.0/0	TCP (6)	443

Added: security group. The security group on the instance in the public subnet allows inbound SSH traffic from anywhere.

Both instances also have the default security group, which allows all outbound connections and any inbound connections from other instances in the same security group.



Network ACLs

- The first rule allows HTTPS traffic, while the second denies all TCP traffic. If an HTTPS connection were to be initiated, it would be allowed, since rule #100 is evaluated first, and it allows the traffic.

Rule #	Type	Protocol	Port range	Source	Allow or deny
100	HTTPS	TCP	443	0.0.0.0/0	Allow
200	ALL	TCP	ALL	0.0.0.0/0	Deny

Network ACLs

Security group rule allowing outbound HTTPS traffic

Type	Destination	Protocol (number)	Port range
Outbound	0.0.0.0/0	TCP (6)	443

Network ACL rules allowing outbound HTTPS traffic

Outbound/inbound	Rule #	Type	Protocol	Port range	Destination	Allow or deny
Outbound	100	HTTPS	TCP	443	0.0.0.0/0	Allow
Outbound	*	ALL	ALL	ALL	0.0.0.0/0	Deny
Inbound	*	ALL	ALL	ALL	0.0.0.0/0	Deny

Network ACLs In Practice

Network ACL rules allowing web traffic originating outside the VPC and traffic to a MongoDB server from within the VPC

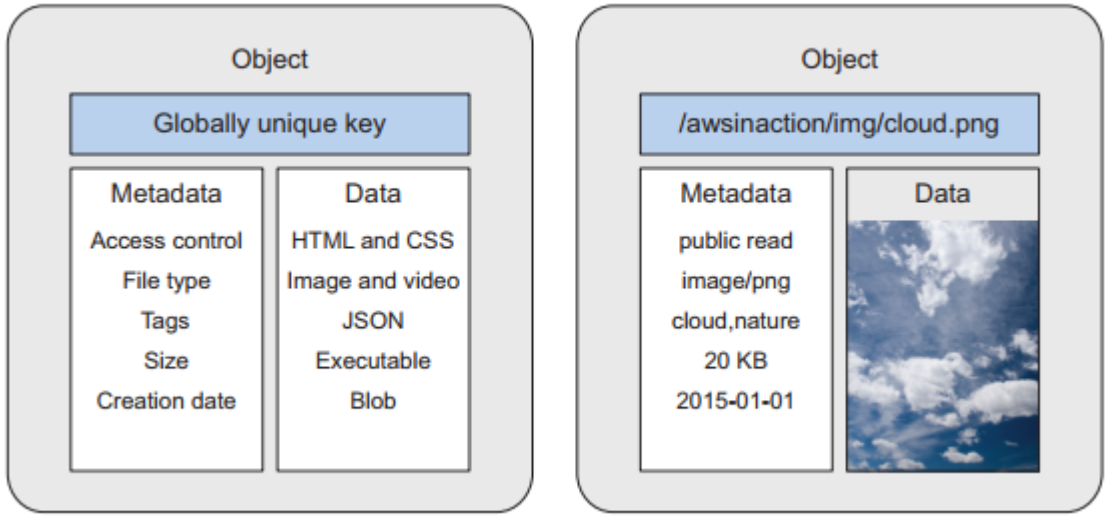
Inbound/ outbound	Rule #	Type	Protocol	Port range	Source/ destination	Allow or deny
Outbound	100	HTTPS	TCP	443	0.0.0.0/0	Allow
Outbound	200	CUSTOM	TCP	27017	10.0.0.0/24	Allow
Outbound	*	ALL	ALL	ALL	0.0.0.0/0	Deny
Inbound	300	HTTPS	TCP	443	0.0.0.0/0	Allow
Inbound	400	CUSTOM	TCP	27017	10.0.0.0/24	Allow
Inbound	*	ALL	ALL	ALL	0.0.0.0/0	Deny

Lesson 7 S3

Michael Yang

What is Object Store?

- Back in the old days, data was managed in a hierarchy consisting of folders and files. The file was the representation of the data. In an *object store*, data is stored as objects. Each object consists of a globally unique identifier, some metadata, and the data itself, as figure illustrates. An object's *globally unique identifier* (GUID) is also known as its *key*; you can address the object from different devices and machines in a distributed system using the GUID.



Amazon S3

- Amazon S3 is a distributed data store, and one of the oldest services provided by AWS. *Amazon S3* is an acronym for *Amazon Simple Storage Service*. It's a typical web service that lets you store and retrieve data organized as objects via an API reachable over HTTPS.

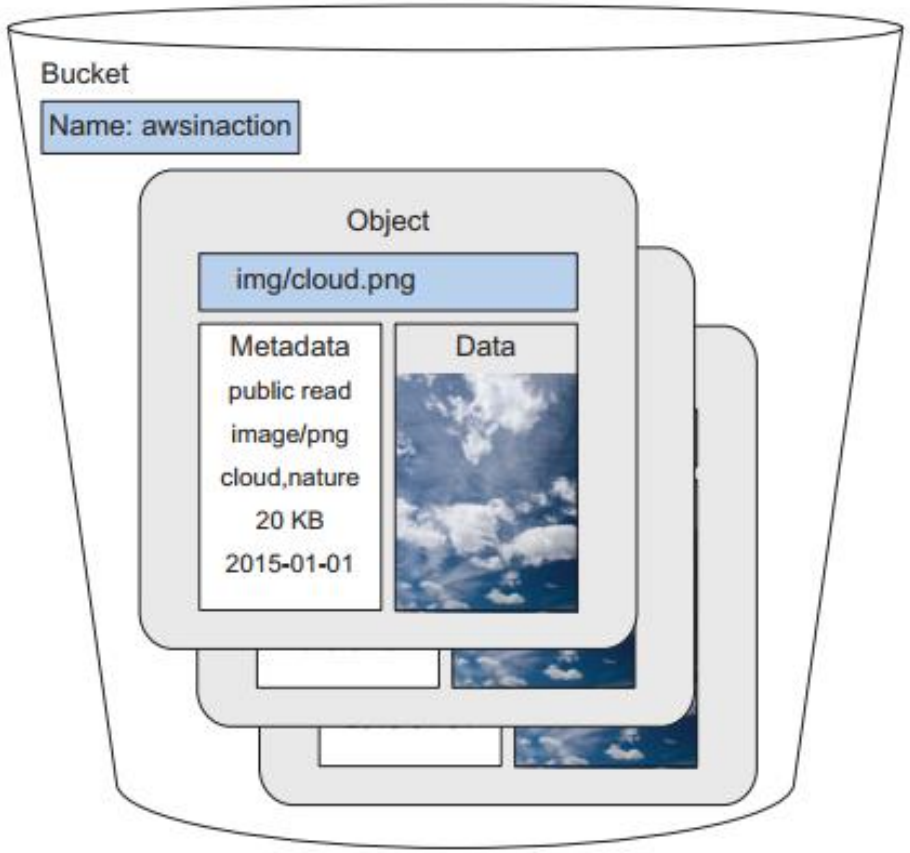
S3 Use Cases

- *Storing and delivering static website content*
- *Backing up data*
- *Storing structured data for analytics, also called a data lake*
- *Storing and delivering user-generated content*

Amazon S3

Uploading and downloading an object to S3 via HTTPS

Bucket



S3 uses buckets with globally unique names to group objects.

Backup with AWS CLI

- Use the AWS CLI to upload data to and download data from S3. This approach isn't limited to offsite backups; you can use it in many other scenarios as well, such as the following
  - Sharing files with your coworkers or partners, especially when working from different locations
  - Storing and retrieving artifacts needed to provision your virtual machines (such as application binaries, libraries, or configuration files)
  - Outsourcing storage capacity to lighten the burden on local storage systems—in particular, for data that is accessed infrequently

AWS CLI for S3

- `$ aws s3 mb s3://$yourbucketname`
- `$ aws s3 sync $path s3://$yourbucketname/backup`
- `$ aws s3 cp --recursive s3://$yourbucketname/backup $path`
- `$ aws s3 rb --force s3://awsinaction-$yourname`

Versioning

- **$ aws s3api put-bucket-versioning --bucket $yourbucketname --versioning-configuration Status=Enabled**

- **$ aws s3api list-object-versions --bucket $yourbucketname**

BucketNotEmpty Error

- If you turn on versioning for your bucket, removing the bucket will cause a `BucketNotEmpty` error. Use the Management Console to delete the bucket in this case as follows:
  1 Open the Management Console with your browser.
  2 Go to the S3 service using the main navigation menu.
  3 Select the bucket you want to delete.
  4 Click the Empty button, and confirm permanently deleting all the objects
  5 Wait until objects and versions have been deleted, and click the Exit button
  6 Select the bucket you want to delete.
  7 Click the Delete button, and confirm deleting the bucket

Optimize your cost

	S3 Standard	S3 Glacier Instant Retrieval	S3 Glacier Flexible Retrieval	S3 Glacier Deep Archive
Storage costs for 1 GB per month in US East (N. Virginia)	$0.023	$0.004	$0.0036	$0.00099
Costs for 1,000 write requests	$0.005	$0.02	$0.03	$0.05
Costs for retrieving data	Low	High	High	Very High
Accessibility	Milliseconds	Milliseconds	1–5 minutes/ 3–5 hours/ 5–12 hours	12 hours/ 48 hours
Durability objective	99.999999999%	99.999999999%	99.999999999%	99.999999999%
Availability objective	99.99%	99.9%	99.99%	99.99%

S3 Glacier

- `$ aws s3 mb s3://$yourbucketname`
- `$ aws s3 cp --storage-class GLACIER $path s3://$yourbucketname`
- `$ aws s3 cp s3://$yourbucketname/$objectkey $path`
- `$ aws s3api restore-object --bucket $yourbucketname --key $objectkey --restore-request Days=1,,GlacierJobParameters={"Tier"="Expedited"}`

Download Archives

```
$ aws s3api head-object --bucket $yourbucketname  --key $objectkey
```

```
{
    "AcceptRanges": "bytes",
    "Expiration": "expiry-date=\"Wed, 12 Jul 2023 ...\", rule-id=\"...\"",
    "Restore": "ongoing-request=\"true\"",
    "LastModified": "2022-07-11T09:26:12+00:00",
    "ContentLength": 112,
    "ETag": "\"c25fa1df1968993d8e647c9dcd352d39\"",
    "ContentType": "binary/octet-stream",
    "Metadata": {},
    "StorageClass": "GLACIER"
}
```

Restoration of the object is still ongoing.

Cleanup

- `$ aws s3 rb --force s3://$yourbucketname`

Storing with SDK

- Listing buckets and their objects
- Creating, removing, updating, and deleting (CRUD) objects and buckets
- Managing access to objects

S3 Integration examples

- *Allow a user to upload a profile picture.* Store the image in S3, and make it publicly accessible. Integrate the image into your website via HTTPS.
- *Generate monthly reports (such as PDFs), and make them accessible to users.* Create the documents and upload them to S3. If users want to download documents, fetch them from S3.
- *Share data between applications.* You can access documents from different applications. For example, application A can write an object with the latest information about sales, and application B can download the document and analyze the data.

Lesson 8 RDS

Michael Yang

RDB and RDS

- Relational databases focus on data consistency and guarantee ACID (atomicity, consistency, isolation, and durability) database transactions. A typical task is storing and querying structured data, such as the accounts and transactions in an accounting application.
- The Amazon Relational Database Service (Amazon RDS) offers ready-to-use relational databases such as PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server.
- RDS is a managed service. The managed service provider—in this case, AWS—is responsible for providing a defined set of services—in this case, operating a relational database system.

RDS vs DB on EC2

Managed service RDS vs. a self-hosted database on virtual machines

	Amazon RDS	Self-hosted on virtual machines
Cost for AWS services	Higher because RDS costs more than virtual machines (EC2)	Lower because virtual machines (EC2) are cheaper than RDS
Total cost of ownership	Lower because operating costs are split among many customers	Much higher because you need your own manpower to manage your database
Quality	AWS professionals are responsible for the managed service.	You'll need to build a team of professionals and implement quality control yourself.
Flexibility	High, because you can choose a relational database system and most of the configuration parameters	Higher, because you can control every part of the relational database system you installed on virtual machines

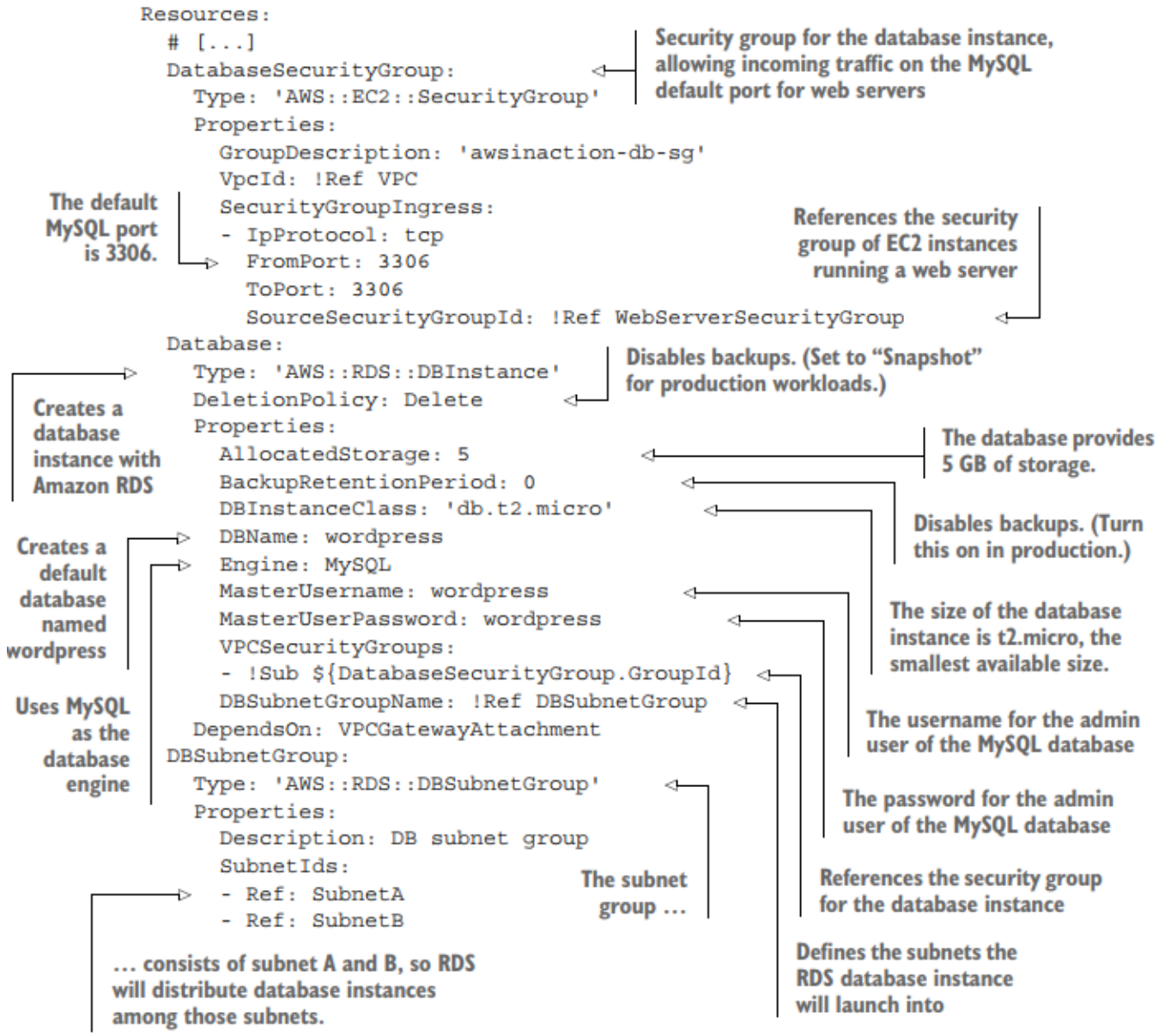
Launch RDS

- Launching a database consists of two steps:
  1 Launching a database instance
  2 Connecting an application to the database endpoint

- Attribute needed to connect to an RDS database

Attribute	Description
`AllocatedStorage`	Storage size of your database in GB
`DBInstanceClass`	Size (also known as instance type) of the underlying virtual machine
`Engine`	Database engine (Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, or Microsoft SQL Server) you want to use
`DBName`	Identifier for the database
`MasterUsername`	Name for the admin user
`MasterUserPassword`	Password for the admin user

CloudFormation Template



- Databases on Amazon RDS are priced according to the size of the underlying virtual machine and the amount and type of allocated storage.

Monthly costs for a medium-sized RDS instance

Description	Monthly price
Database instance `db.t4g.medium`	$94.17 USD
50 GB of general purpose (SSD)	$11.50 USD
Additional storage for database snapshots (100 GB)	$9.50 USD
Total	$115.17 USD

Importing Data

- A database without data isn't useful. In many cases, you'll need to import data into a new database by importing a dump from the old database. The process is similar for all other database engines (Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server). Here we learn through the process of importing a MySQL database dump to an RDS database with a MySQL engine.

Importing Data

- To import a database from your on-premises environment to Amazon RDS, follow these steps:

1. Export the database. (using mysqldump)
2. Start a virtual machine in the same region and VPC as the RDS database.
3. Upload the database dump to the virtual machine.
4. Run an import of the database dump to the RDS database on the virtual server.

Importing Data

- Theoretically, you could import a database to RDS from any machine from your on-premises or local network, but the higher latency over the internet or VPN connection will slow down the import process dramatically. Because of this, we recommend adding a second step: upload the database dump to a virtual machine running in the same AWS region and VPC, and import the database into RDS from there.

Importing Data

- To do so, we'll guide you through the following steps:

1 Connect to the virtual machine that is running.
2 Download a database dump from S3 to the VM. (If you are using your own database dump, we recommend uploading it to S3 first.)
3 Import the database dump into the RDS database from the virtual machine.

Backup and Restore

- How to use RDS snapshots to do backup and restore
  1. Configuring the retention period and time frame for automated snapshots
  2. Creating snapshots manually
  3. Restoring snapshots by starting new database instances based on a snapshot
  4. Copying a snapshot to another region for disaster recovery or relocation

Configure Automated Snapshot

- BackupRetentionPeriod
- Automated snapshots are created once a day during the specified time frame. If no time frame is specified, RDS picks a random 30-minute time frame during the night. A new random time frame will be chosen each night.

Configure Automated Snapshot

```
aws cloudformation update-stack --stack-name wordpress --template-url \
 https://s3.amazonaws.com/awsinaction-code3/chapter10/\
 template-snapshot.yaml \
 --parameters ParameterKey=WordpressAdminPassword,UsePreviousValue=true \
 --capabilities CAPABILITY_IAM
```

Configure Automated Snapshot

```
Database:
  Type: 'AWS::RDS::DBInstance'
  DeletionPolicy: Delete
  Properties:
    AllocatedStorage: 5
    BackupRetentionPeriod: 3
    PreferredBackupWindow: '05:00-06:00'
    DBInstanceClass: 'db.t2.micro'
    DBName: wordpress
    Engine: MySQL
    MasterUsername: wordpress
    MasterUserPassword: wordpress
    VPCSecurityGroups:
    - !Sub ${DatabaseSecurityGroup.GroupId}
    DBSubnetGroupName: !Ref DBSubnetGroup
  DependsOn: VPCGatewayAttachment
```

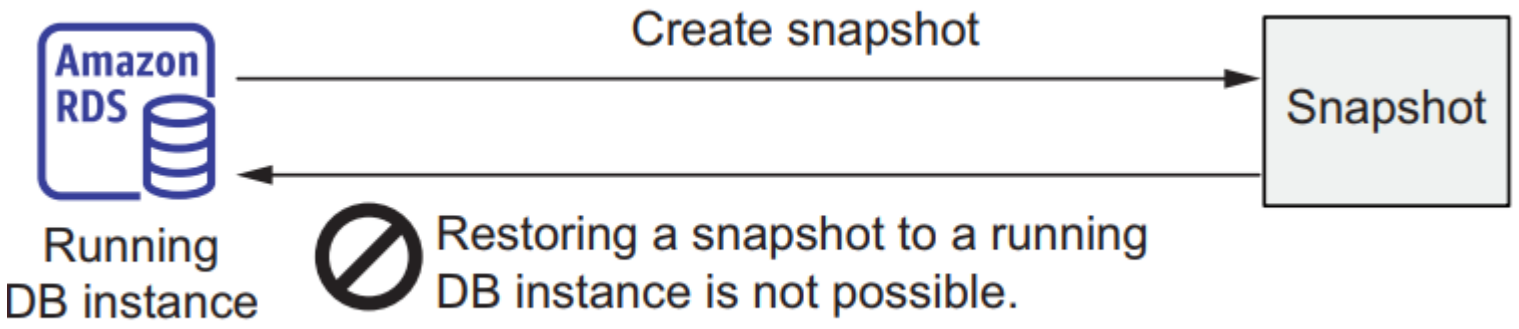
Keeps snapshots for three days

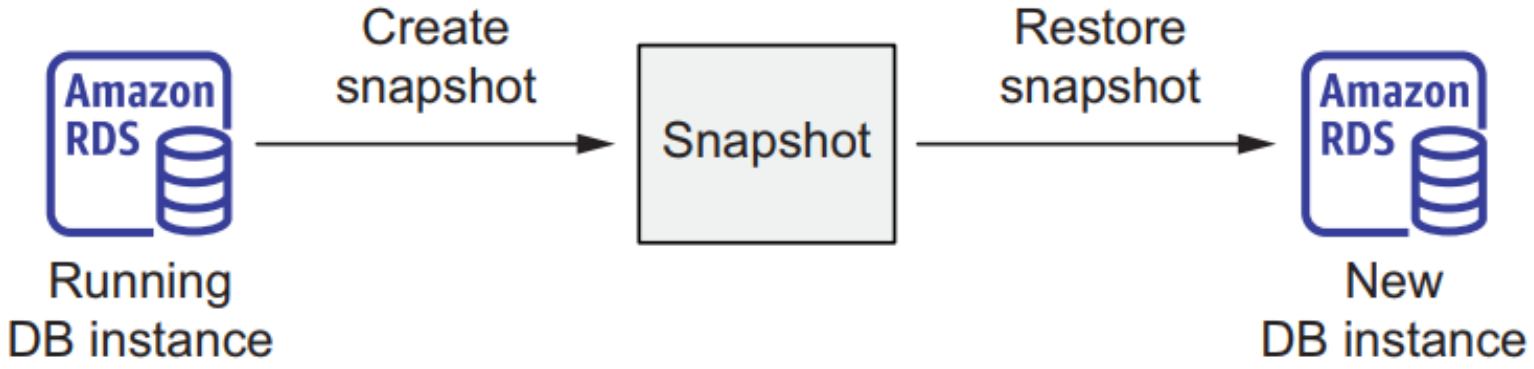
Creates snapshots automatically between 05:00 and 06:00 UTC

Backup Manually

- To create a snapshot, you have to know the instance identifier. The following command extracts the instance identifier from the first RDS database instance:
  `$ aws rds describe-db-instances --output text --query "DBInstances[0].DBInstanceIdentifier"`
- The next command creates a manual snapshot called `wordpress-manual-snapshot`. Replace `$DBInstanceIdentifier` with the output of the previous command:
  `$ aws rds create-db-snapshot --db-snapshot-identifier wordpress-manual-snapshot --db-instance-identifier $DBInstanceIdentifier`

Restoring Database





Restoring Database

- To create a new database in the same VPC as the WordPress platform you started in, you need to find out the existing database's subnet group.

```
$ aws cloudformation describe-stack-resource \
➥ --stack-name wordpress --logical-resource-id DBSubnetGroup \
➥ --query "StackResourceDetail.PhysicalResourceId" --output text
```

- You're now ready to create a new database based on the manual snapshot you created at the beginning of this section. Execute the following command, replacing `$SubnetGroup` with the output of the previous command:

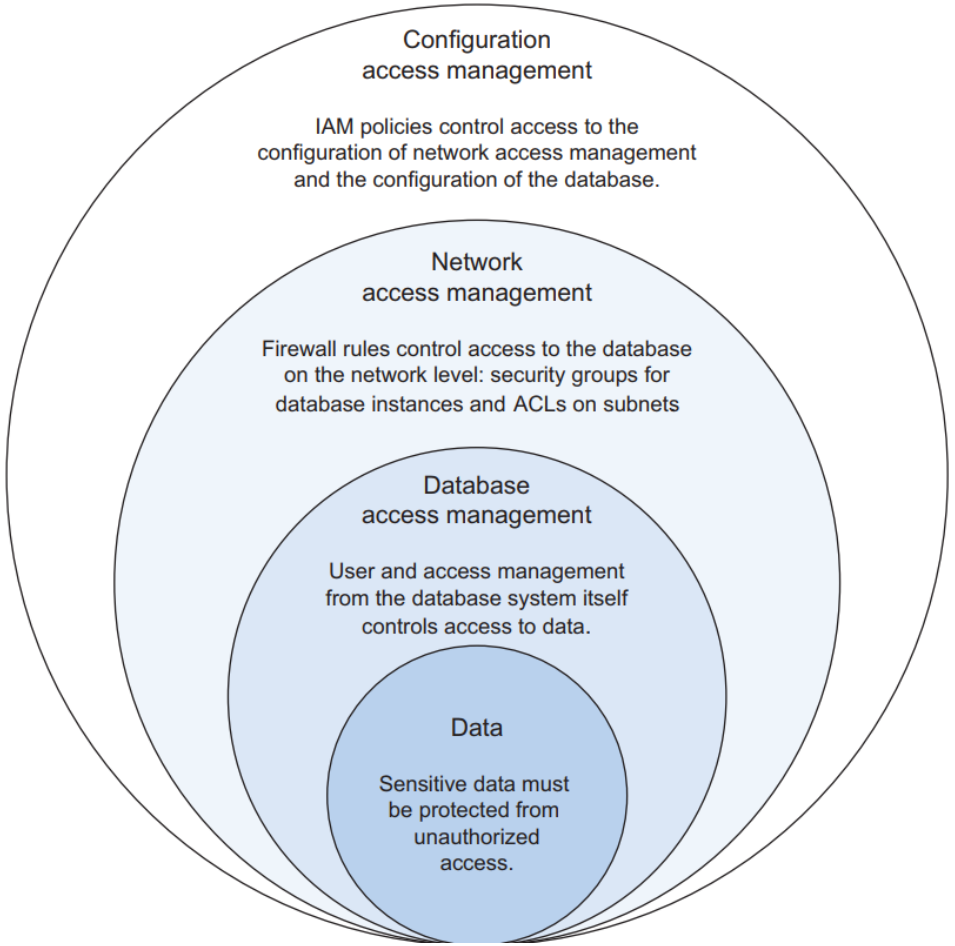
```
$ aws rds restore-db-instance-from-db-snapshot \
➥ --db-instance-identifier awsinaction-db-restore \
➥ --db-snapshot-identifier wordpress-manual-snapshot \
➥ --db-subnet-group-name $SubnetGroup
```

Restoring Automated Snapshot

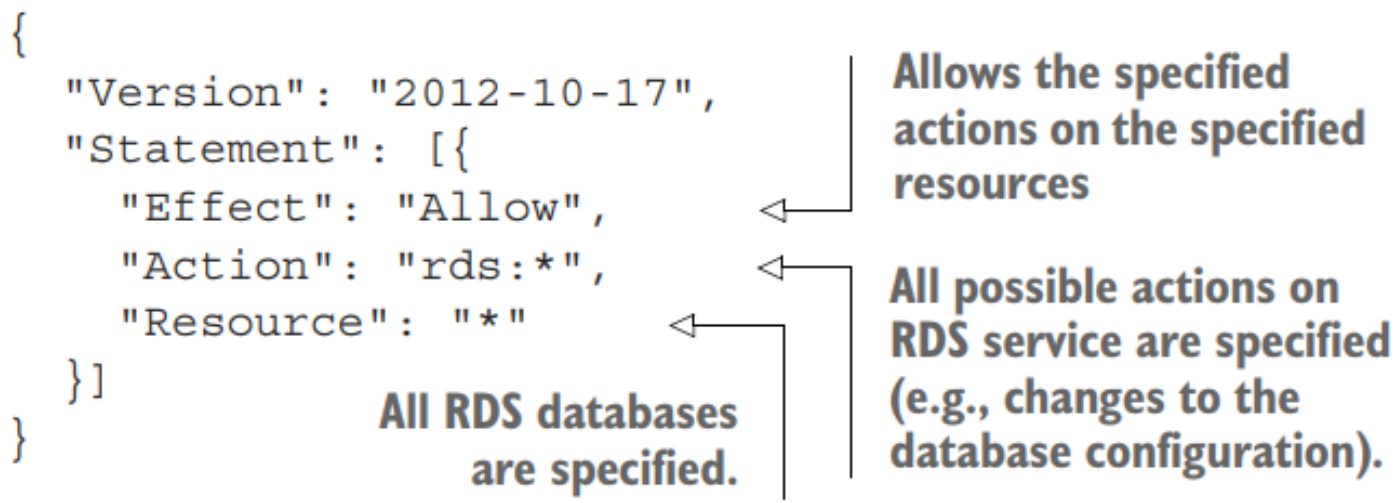
- If you're using automated snapshots, you can also restore your database from a specified moment, because RDS keeps the database's change logs. This allows you to jump back to any point in time from the backup retention period to the last five minutes.

```
$ aws rds restore-db-instance-to-point-in-time \
➥ --target-db-instance-identifier awsinaction-db-restore-time \
➥ --source-db-instance-identifier $DBInstanceIdentifier \
➥ --restore-time $Time --db-subnet-group-name $SubnetGroup
```

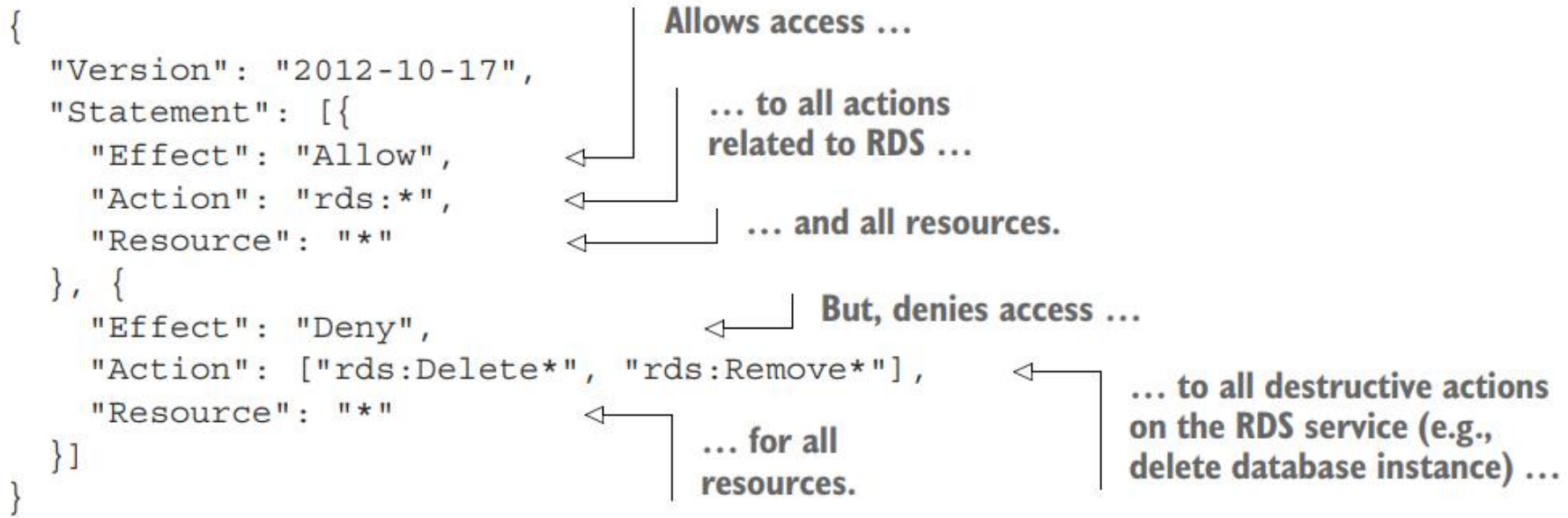
Controlling Access to RDS



Controlling Persmissons



Controlling Permissions



Controlling Network

```
DatabaseSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'awsinaction-db-sg'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: 3306
      ToPort: 3306
      SourceSecurityGroupId: !Ref WebServerSecurityGroup
```

The security group for the database instance, allowing incoming traffic on the MySQL default port for web servers

The default MySQL port is 3306.

References the security group for web servers

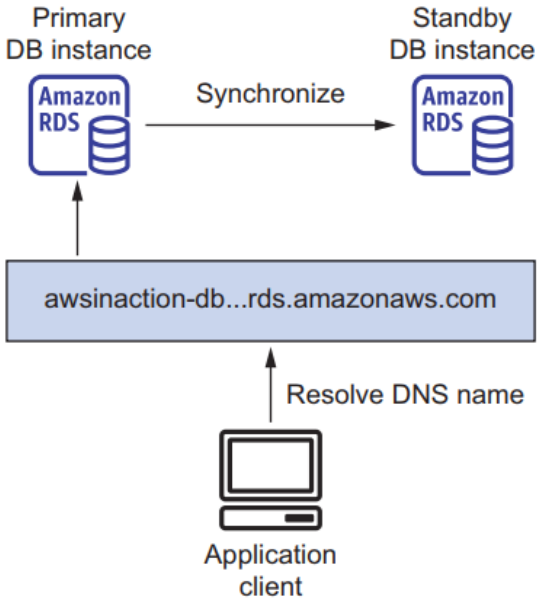
Controlling Data Access

- A database engine also implements access control itself. User management of the database engine has nothing to do with IAM users and access rights; it's only responsible for controlling access to the database. For example, you typically define a user for each application and grant rights to access and manipulate tables as needed. In the WordPress example, a database user called wordpress is created. The WordPress application authenticates itself to the database engine (MySQL, in this case) with this database user and a password.

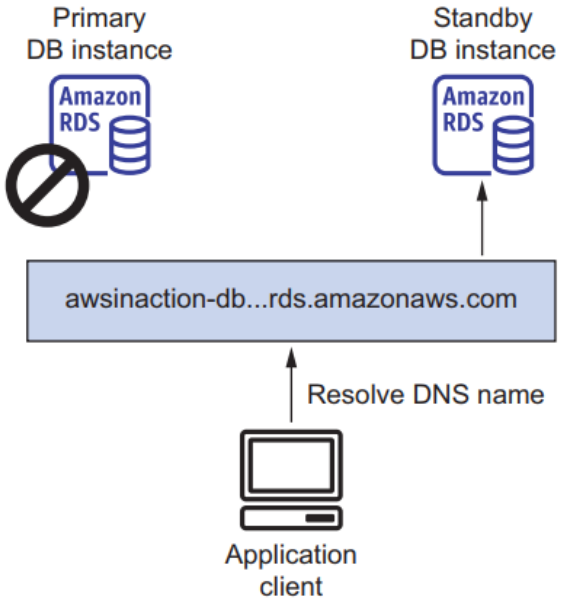
Controlling Data Access

- Typical use cases follow:
  1Limiting write access to a few database users (e.g., only for an application)
  2 Limiting access to specific tables to a few users (e.g., to one department in the organization)
  3 Limiting access to tables to isolate different applications (e.g., hosting multiple applications for different customers on the same database)

HA Database



The primary database is replicated to the standby database when running in high-availability mode.



The client fails over to the standby database if the primary database fails, using DNS resolution.

Aurora

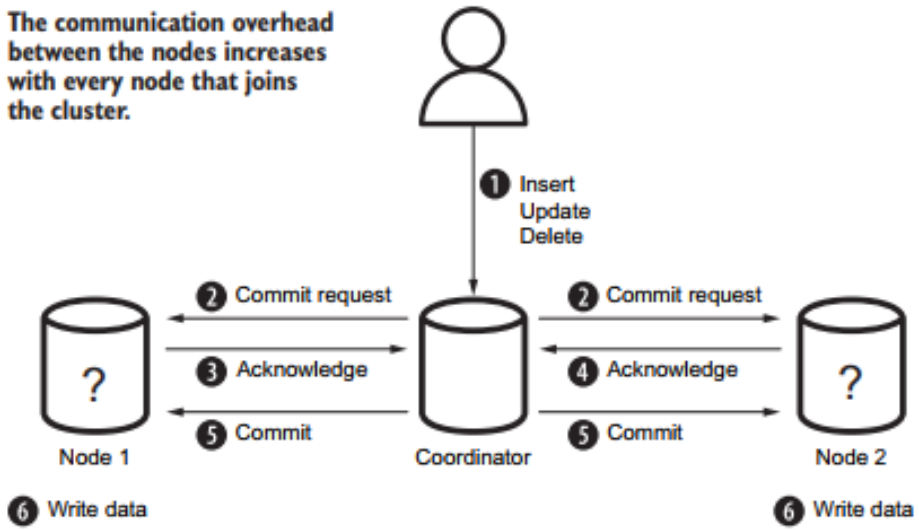
- Aurora is an exception to the way that highly available databases operate in AWS. It does not store your data on a single EBS volume. Instead, Aurora stores data on a cluster volume. A cluster volume consists of multiple disks, with each disk having a copy of the cluster data. This implies that the storage layer of Aurora is not a single point of failure. But still, only the primary Aurora database instance accepts write requests. If the primary goes down, it is automatically re-created, which typically takes less than 10 minutes. If you have replica instances in your Aurora cluster, a replica is promoted to be the new primary instance, which usually takes around one minute and is much faster than primary re-creation.

Lesson 9 DynamoDB

Michael Yang

Why NoSQL?

- *Vertically*—You can use faster hardware for your database machine; for example, you can add memory or replace the CPU with a more powerful model. *Horizontally*—You can add a second database machine. Both machines then form a *database cluster*.
- Relationanl Database, two phase commit



What is DynamoDB

- There are four types of NoSQL databases—document, graph, columnar, and key-value store—each with its own uses and applications. Amazon provides a NoSQL database service called *DynamoDB*, a key-value store.
- DynamoDB is a fully managed, proprietary, closed source key-value store with document support. In other words, DynamoDB persists objects identified by a unique key, which you might know from the concept of a hash table.
- DynamoDB is highly available and highly durable. You can scale from one item to billions and from one request per second to tens of thousands of requests per second.

Use Cases

- Some typical use cases for DynamoDB follow:

1. When building systems that need to deal with a massive amount of requests or spiky workloads, the ability to scale horizontally is a game changer. We have used DynamoDB to track client-side errors from a web application, for example.

2. When building small applications with a simple data structure, the pay-perrequest pricing model and the simplicity of a fully managed service are good reasons to go with DynamoDB. For example, we used DynamoDB to track the progress of batch jobs.

Example – A To-do-list Application

- “nodetodo” uses DynamoDB as a database and comes with the following features:

1 Creates and deletes users
2 Creates and deletes tasks
3 Marks tasks as done
4 Gets a list of all tasks with various filters

- You’ll implement these commands for “nodetodo” app. This listing shows the full CLI description of all the commands, including parameters.

```
nodetodo

Usage:
  nodetodo user-add <uid> <email> <phone>
  nodetodo user-rm <uid>
  nodetodo user-ls [--limit=<limit>] [--next=<id>]
  nodetodo user <uid>
  nodetodo task-add <uid> <description> \
➥ [<category>] [--dueat=<yyyymmdd>]
  nodetodo task-rm <uid> <tid>
  nodetodo task-ls <uid> [<category>] \
➥ [--overdue|--due|--withoutdue|--futuredue]
  nodetodo task-la <category> \
➥ [--overdue|--due|--withoutdue|--futuredue]
  nodetodo task-done <uid> <tid>
  nodetodo -h | --help
  nodetodo --version

Options:
  -h --help         Show this screen.
  --version         Show version.
```

The named parameters limit and next are optional.

The category parameter is optional.

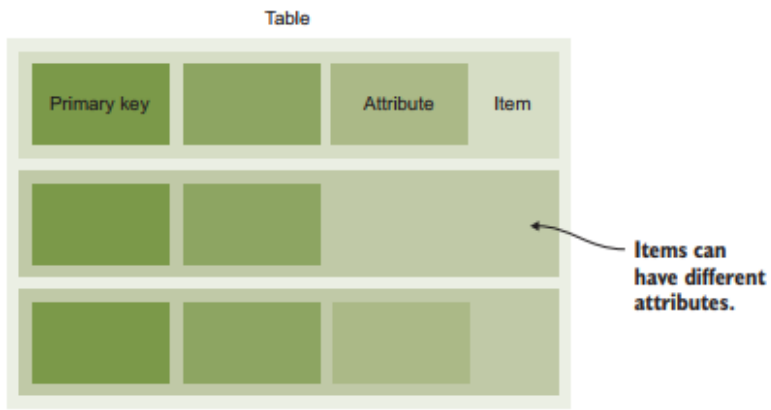
Pipe indicates either/or.

Version information

help prints information about how to use nodetodo.

Example- Creating Tables

- An *item* is a collection of attributes
- an *attribute* is a name-value pair. The attribute value can be scalar (number, string, binary, Boolean), multivalued (number set, string set, binary set), or a JSON document (object, array). Items in a table aren't required to have the same attributes; there is no enforced schema.
- DynamoDB doesn't need a static schema like a relational database does, but you must define the attributes that are used as the primary key in your table. In other words, you must define the table's primary key schema.



DynamoDB tables store items consisting of attributes identified by a primary key.

“Users” Table

- Table name is “todo-user”
- A primary key consists of **one or two** attributes. A primary key is unique within a table and identifies an item.
- When using a single attribute as primary key, DynamoDB calls this the ***partition key*** of the table.

Create DynamoDB table

- The `aws dynamodb create-table` command has the following four mandatory options:

`table-name`—Name of the table (can't be changed)

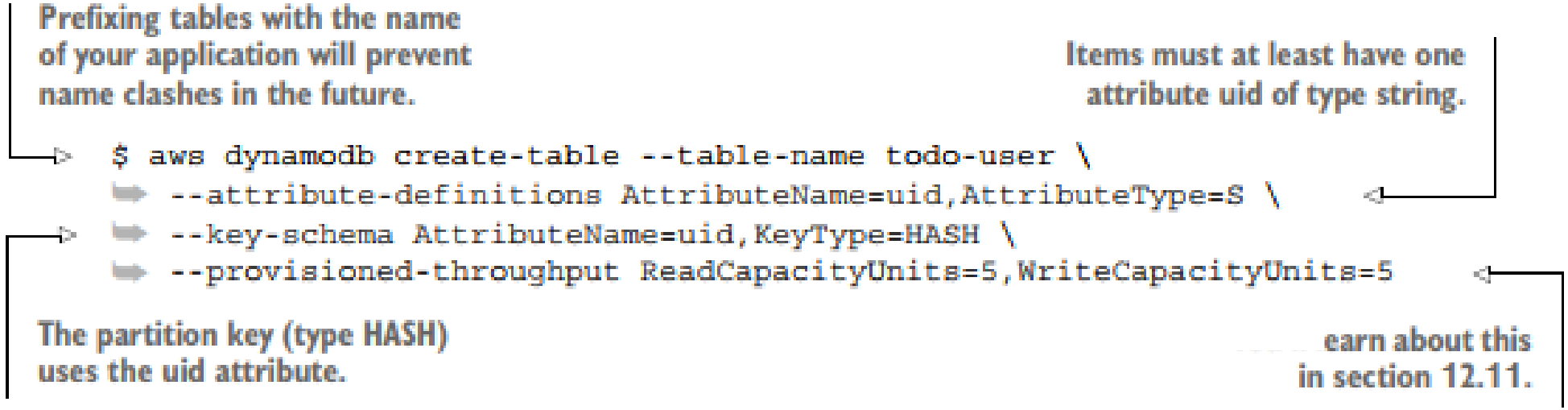
`attribute-definitions`—Name and type of attributes used as the primary key. Multiple definitions can be given using the syntax `AttributeName=attr1,AttributeType=S`, separated by a space character. Valid types are `S` (string), `N` (number), and `B` (binary).

`key-schema`—Name of attributes that are part of the primary key (can't be changed). Contains a single entry using the syntax `AttributeName=attr1, KeyType=HASH` for a partition key, or two entries separated by spaces for a partition key and sort key. Valid types are `HASH` and `RANGE`.

`provisioned-throughput`—Performance settings for this table defined as `ReadCapacityUnits=5,WriteCapacityUnits=5` (you'll learn about this later).

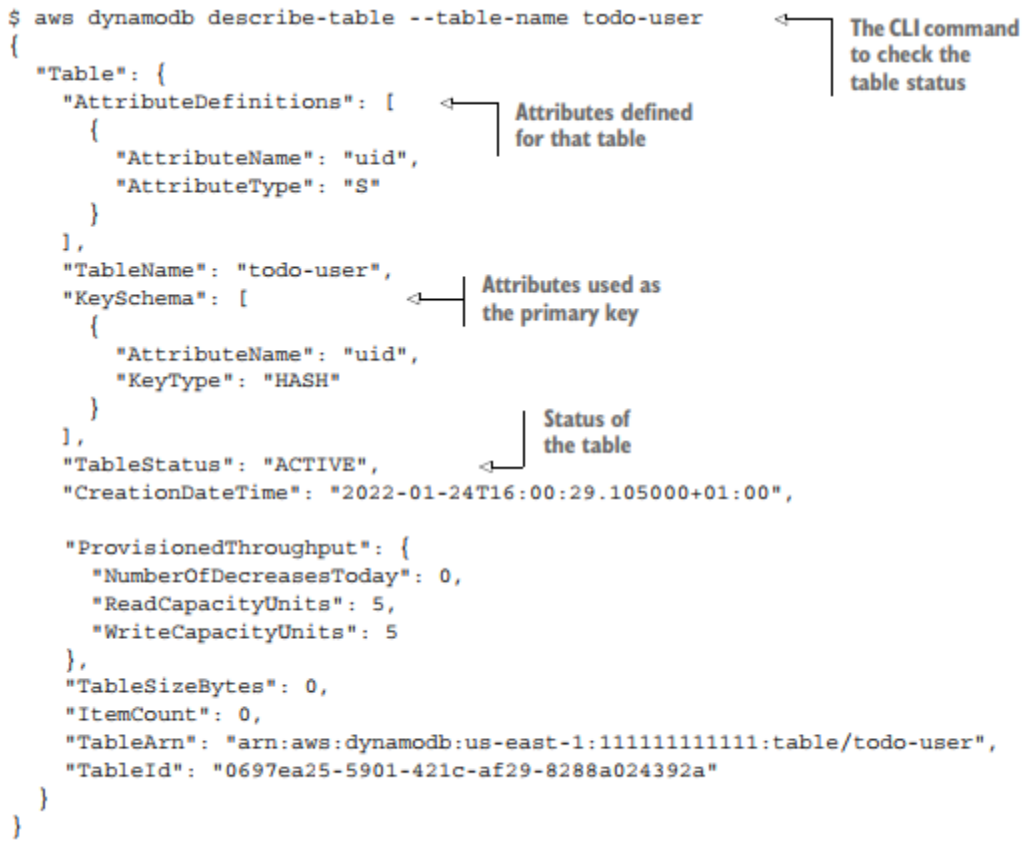
AWS CLI – create-table

- Execute the following command to create the `todo-user` table with the `uid` attribute as the partition key:

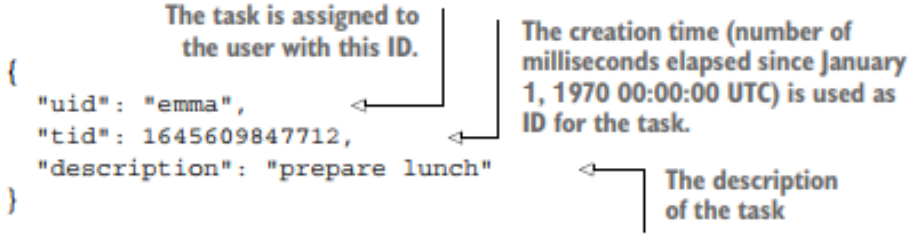


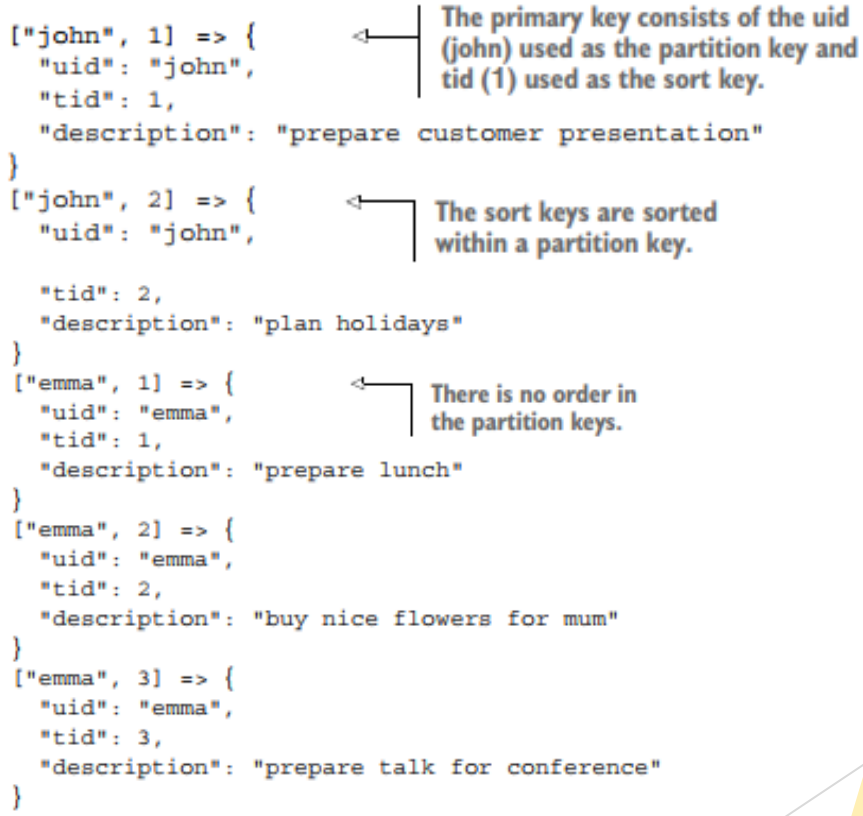
AWS CLI – check status

- Creating a table takes some time. Wait until the status changes to `ACTIVE`. You can check the status of a table as follows



“Tasks” Table


The task is assigned to the user with this ID.
The creation time (number of milliseconds elapsed since January 1, 1970 00:00:00 UTC) is used as ID for the task.
The description of the task
{
"uid": "emma",
"tid": 1645609847712,
"description": "prepare lunch"
}


The primary key consists of the uid (john) used as the partition key and tid (1) used as the sort key.
The sort keys are sorted within a partition key.
There is no order in the partition keys.
["john", 1] => {
"uid": "john",
"tid": 1,
"description": "prepare customer presentation"
}
["john", 2] => {
"uid": "john",
"tid": 2,
"description": "plan holidays"
}
["emma", 1] => {
"uid": "emma",
"tid": 1,
"description": "prepare lunch"
}
["emma", 2] => {
"uid": "emma",
"tid": 2,
"description": "buy nice flowers for mum"
}
["emma", 3] => {
"uid": "emma",
"tid": 3,
"description": "prepare talk for conference"
}

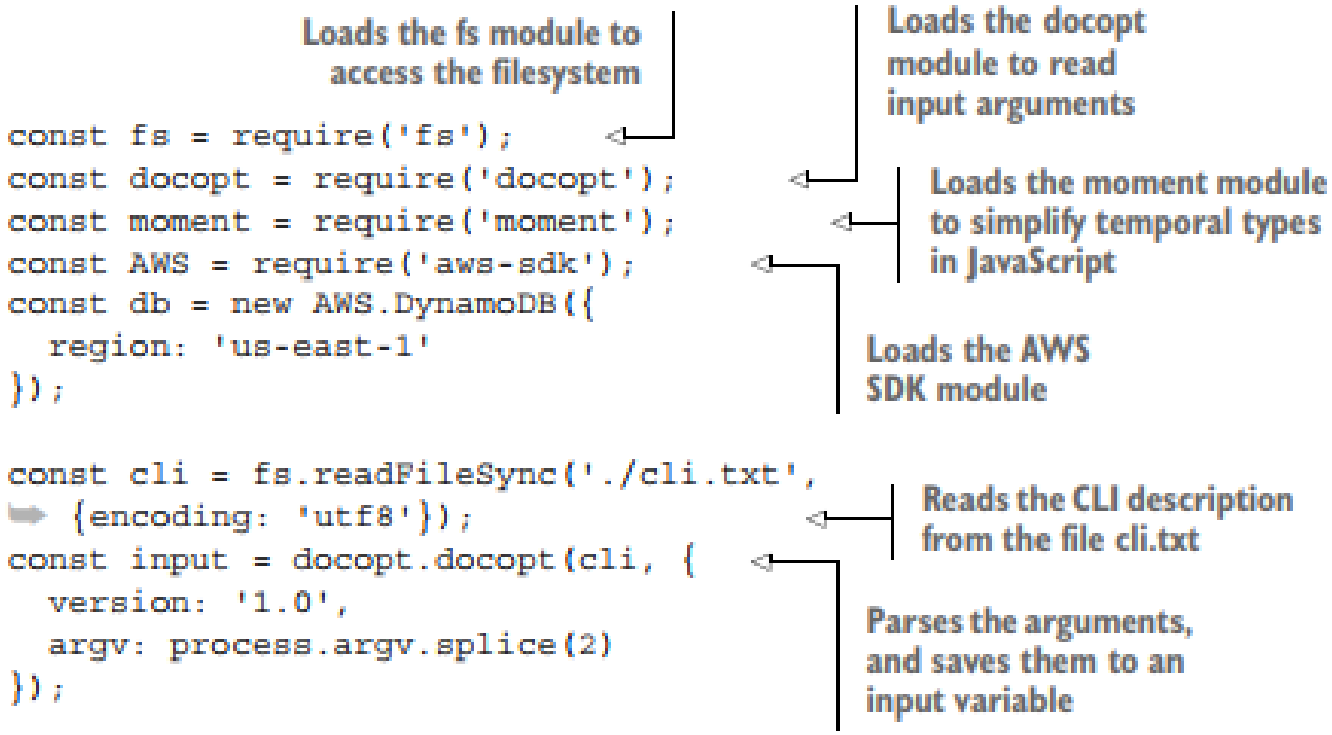
AWS CLI – create table

At least two attributes are needed for a partition key and sort key.

```
$ aws dynamodb create-table --table-name todo-task \
    --attribute-definitions AttributeName=uid,AttributeType=S \
    AttributeName=tid,AttributeType=N \
    --key-schema AttributeName=uid,KeyType=HASH \
    AttributeName=tid,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

The tid attribute is the sort key.

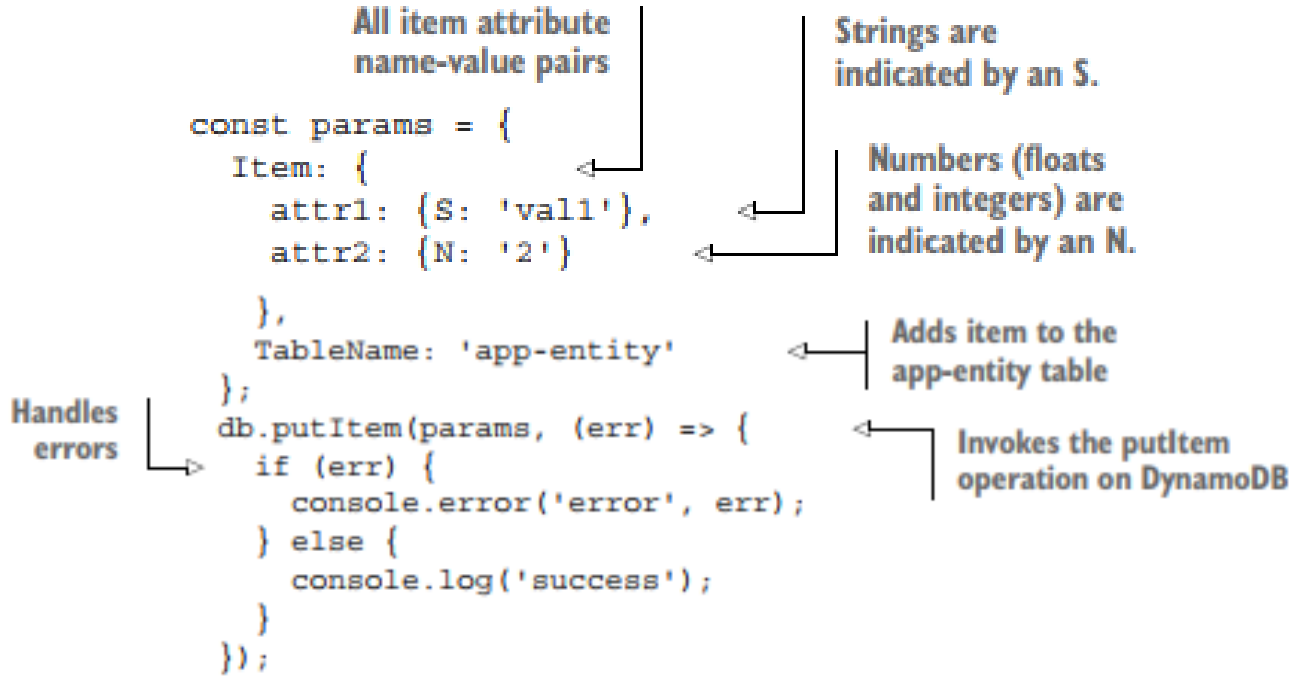
Programming to add items



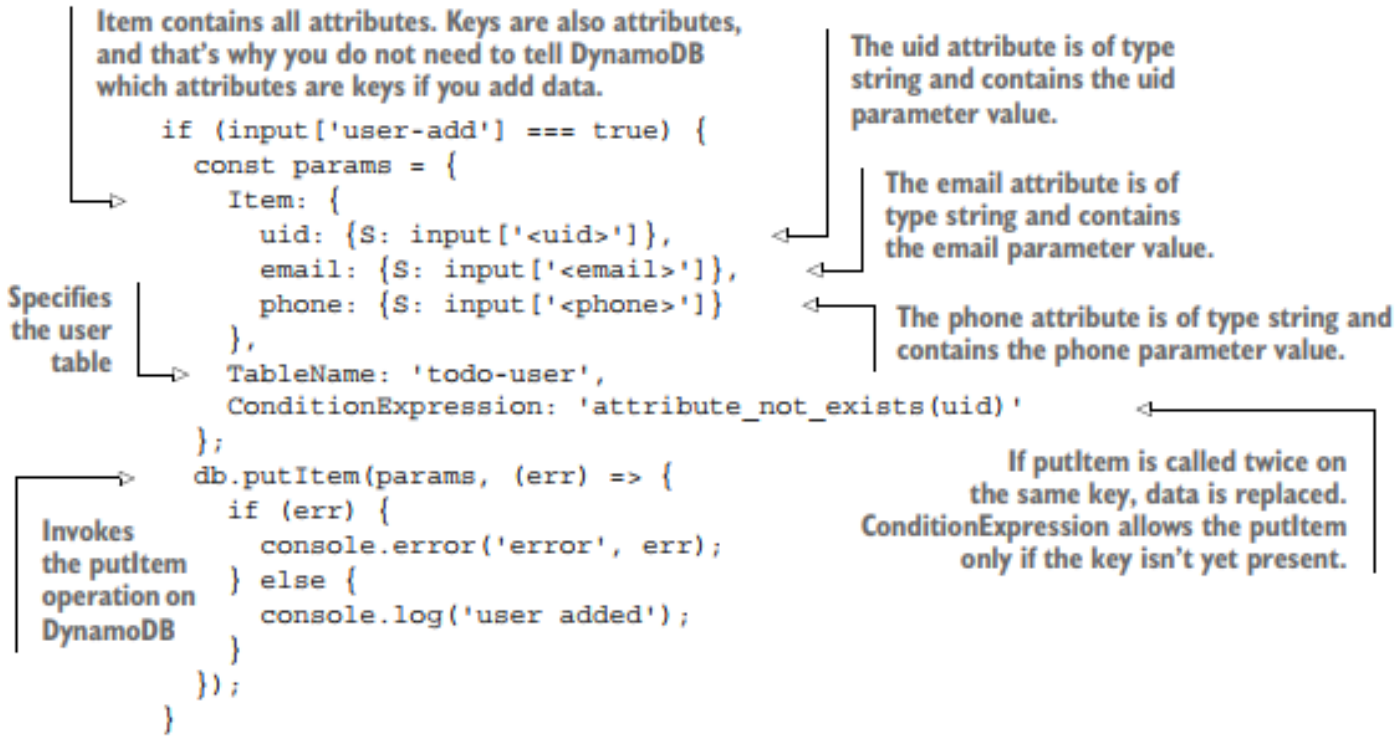

```
const fs = require('fs');
const docopt = require('docopt');
const moment = require('moment');
const AWS = require('aws-sdk');
const db = new AWS.DynamoDB({
  region: 'us-east-1'
});

const cli = fs.readFileSync('./cli.txt',
➥ {encoding: 'utf8'});
const input = docopt.docopt(cli, {
  version: '1.0',
  argv: process.argv.splice(2)
});
```

AWS SDK – How to add items



AWS SDK – Add items to “users” table

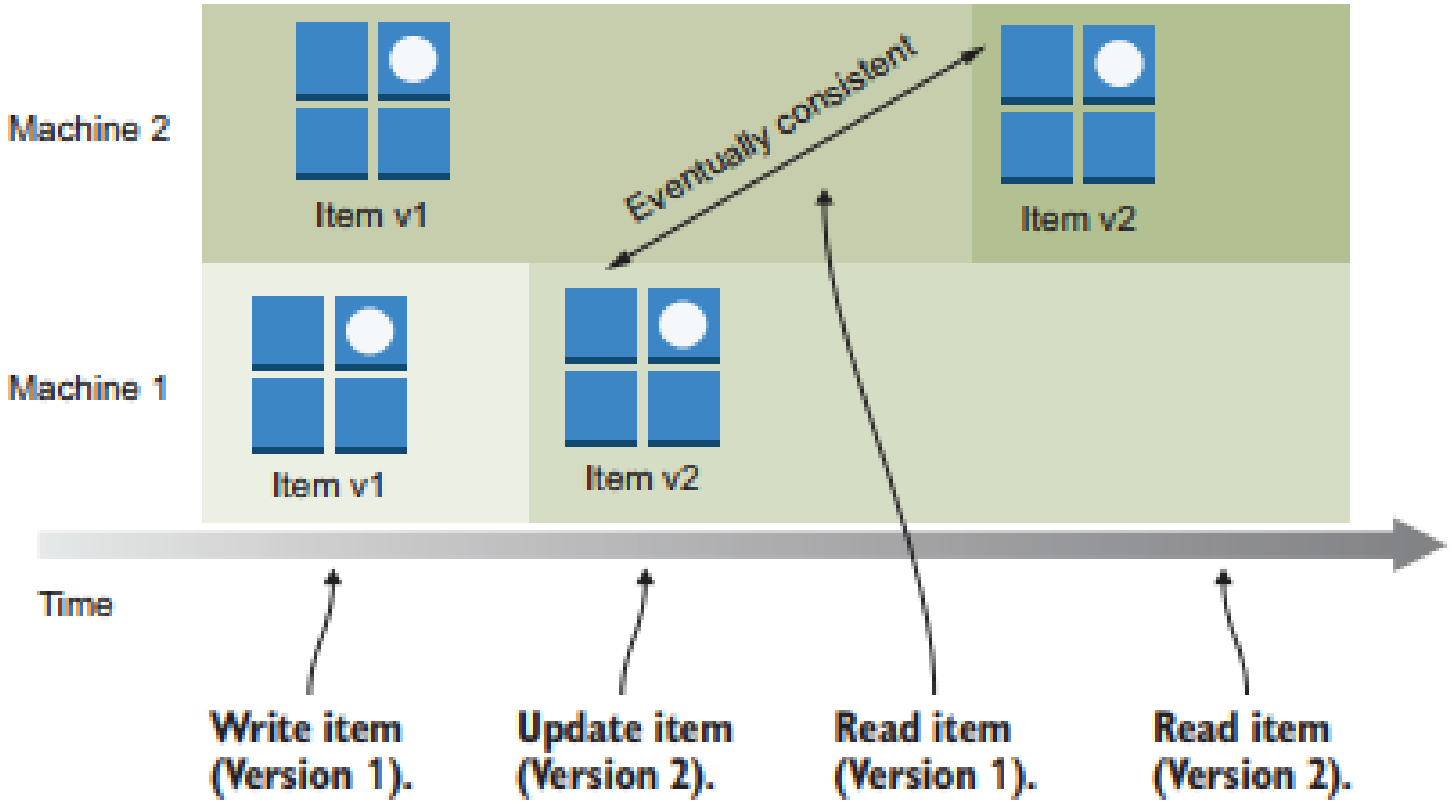



```
if (input['user-add'] === true) {
  const params = {
    Item: {
      uid: {S: input['<uid>']},
      email: {S: input['<email>']},
      phone: {S: input['<phone>']}
    },
    TableName: 'todo-user',
    ConditionExpression: 'attribute_not_exists(uid)'
  };
  db.putItem(params, (err) => {
    if (err) {
      console.error('error', err);
    } else {
      console.log('user added');
    }
  });
}
```

Execute the following commands to add two users:

```
node index.js user-add john john@widdix.de +11111111
node index.js user-add emma emma@widdix.de +22222222
```

Eventually consistent



Eventually consistent reads can return old values after a write operation until the change is propagated to all machines.

PartiQL

The command execute-statement supports PartiQL statements as well.

```
$ aws dynamo3db execute-statement \
➥ --statement "SELECT * FROM \""todo-task\""
```

A simple SELECT statement to fetch all attributes of all items from table todo-task. The escaped " is required because the table name includes a hyphen.

```
$ aws dynamodb execute-statement --statement \
➥ "SELECT * FROM \""todo-task\".\"category-index\""
➥ WHERE category = 'shopping'"
```

```
aws dynamodb execute-statement --statement \
➥ "Update \"todo-user\" SET phone='+33333333' WHERE uid='emma'"
```

DynamoDB Local

- Don’t run DynamoDB Local in production! It’s only made for development purposes and provides the same functionality as DynamoDB, but it uses a different implementation: only the API is the same.

- If you looking for a graphical user interface to interact with DynamoDB, Check out NoSQL Workbench for DynamoDB at http://mng.bz/mJ5P. The tool allows you to create data models, analyze data, and import and export data.

Operating

- DynamoDB isn’t software you can download. Instead, it’s a NoSQL database as a service.
- DynamoDB runs on a fleet of machines operated by AWS.
- DynamoDB replicates your data among multiple machines and across multiple data centers.

Scaling

- a DynamoDB table has two different read/write capacity modes:

On-demand mode adapts the read and write capacity automatically.

Provisioned mode requires you to configure the read and write capacity upfront.

Comparing pricing of DynamoDB on-demand and provisioned mode

Throughput	On-demand mode	Provisioned mode
10 writes per second	$32.85 per month	$4.68 per month
100 reads per second	$32.85 per month	$4.68 per month

Scaling

- Capacity Units

```
$ aws dynamodb get-item --table-name todo-user \
   --key '{"uid": {"S": "emma"}}' \
   --return-consumed-capacity TOTAL \
   --query "ConsumedCapacity"
{
    "CapacityUnits": 0.5,
    "TableName": "todo-user"
}
```

Tells DynamoDB to return the used capacity units

getItem requires 0.5 capacity units.

```
$ aws dynamodb get-item --table-name todo-user \
   --key '{"uid": {"S": "emma"}}' \
   --consistent-read --return-consumed-capacity TOTAL \
   --query "ConsumedCapacity"
{
    "CapacityUnits": 1.0,
    "TableName": "todo-user"
}
```

A consistent read ...

... needs twice as many capacity units.

Networking

- DynamoDB does not run in your VPC. It is accessible via an API. You need internet connectivity to reach the DynamoDB API.