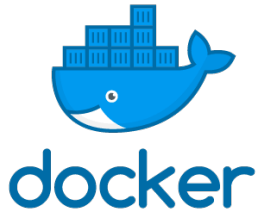
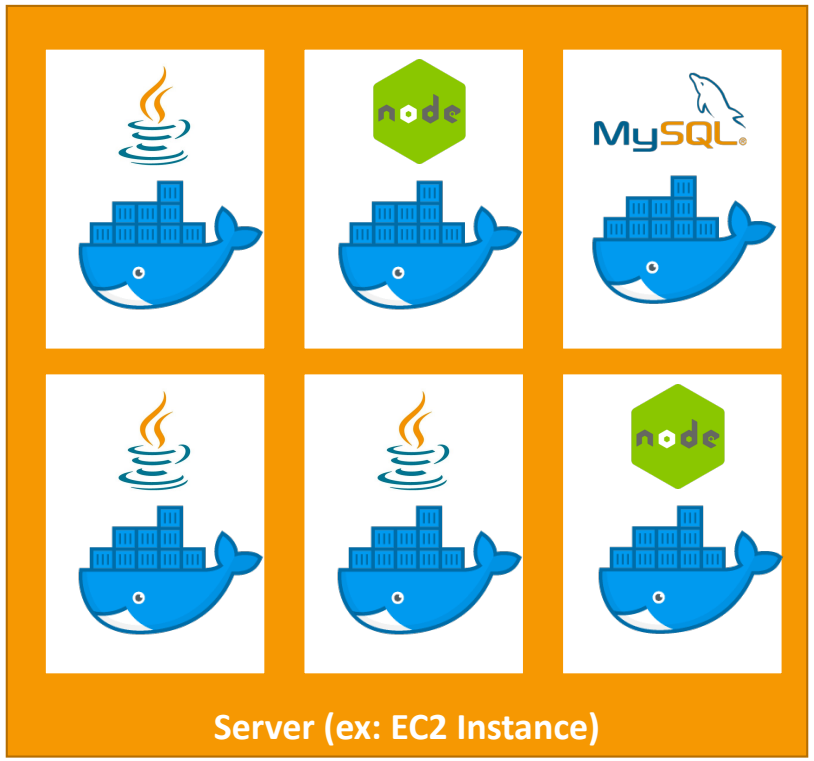
Other Compute Section

What is Docker?



- Docker is a software development platform to deploy apps
- Apps are packaged in **containers** that can be run on any OS
- **Apps run the same, regardless of where they're run**
  - Any machine
  - No compatibility issues
  - Predictable behavior
  - Less work
  - Easier to maintain and deploy
  - Works with any language, any OS, any technology
- Scale containers up and down very quickly (seconds)

Docker on an OS



Where Docker images are stored?

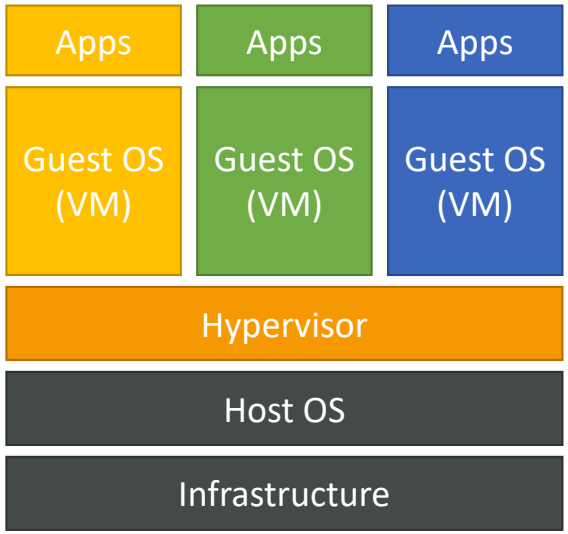
- Docker images are stored in Docker Repositories

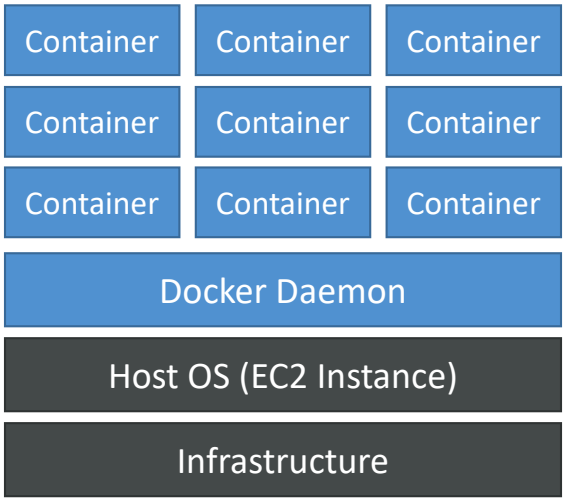
- Public: Docker Hub https://hub.docker.com/
  - Find base images for many technologies or OS:
  - Ubuntu
  - MySQL
  - NodeJS, Java…

- Private: Amazon ECR (Elastic Container Registry)

Docker versus Virtual Machines

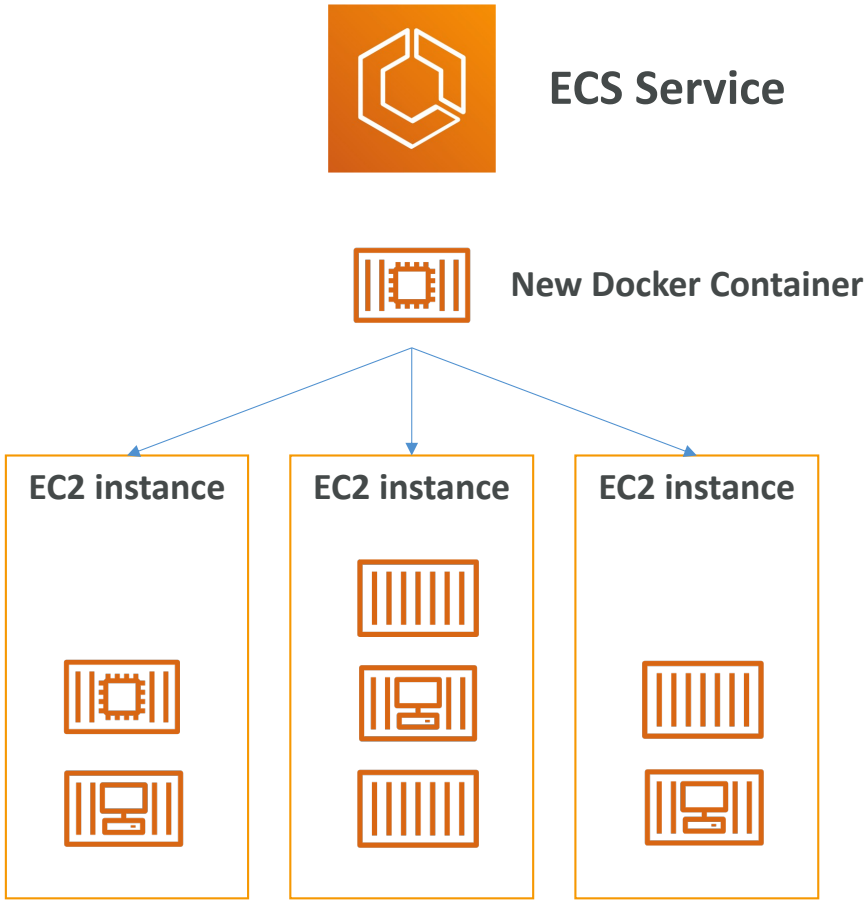
- Docker is "sort of" a virtualization technology, but not exactly
- Resources are shared with the host => many containers on one server





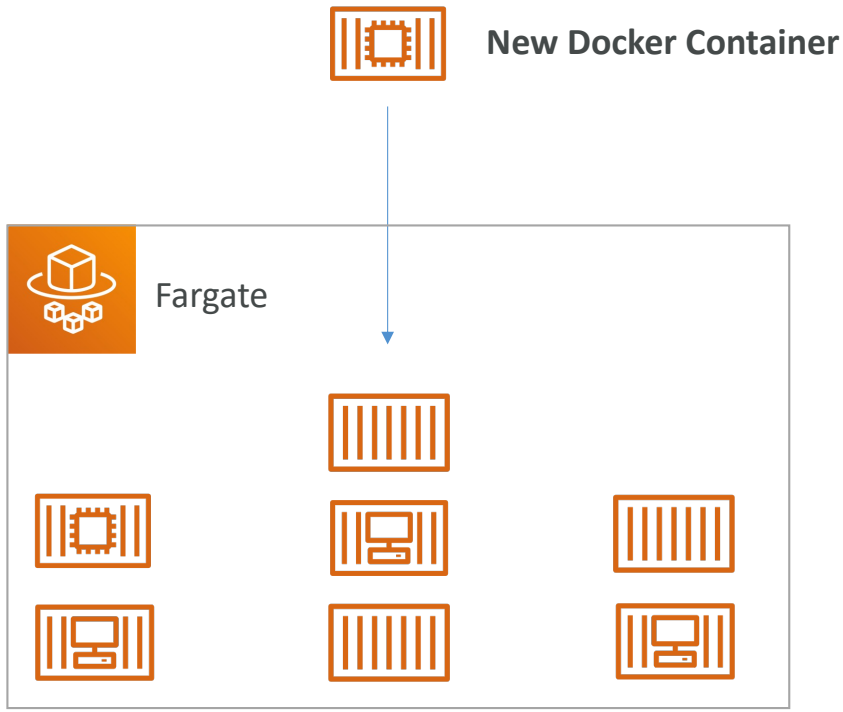
ECS

- ECS = Elastic Container Service
- Launch Docker containers on AWS
- You must provision & maintain the infrastructure (the EC2 instances)
- AWS takes care of starting / stopping containers
- Has integrations with the Application Load Balancer



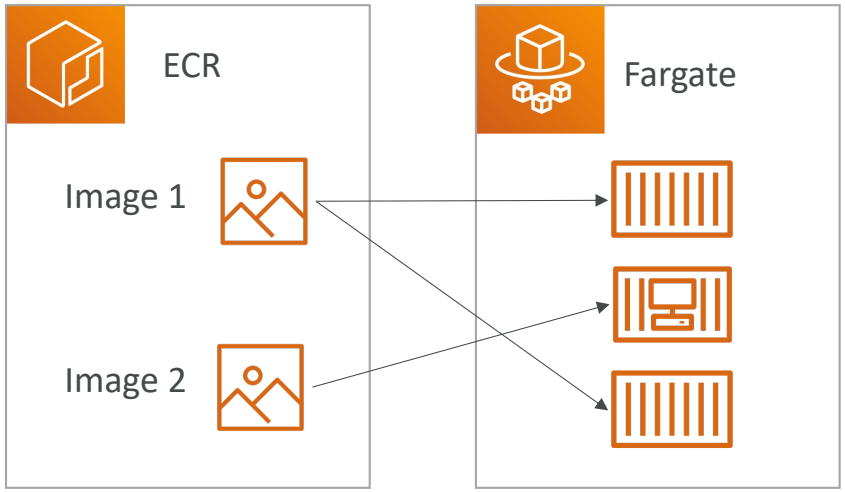
Fargate

- Launch Docker containers on AWS
- You do not provision the infrastructure (no EC2 instances to manage) – simpler!
- Serverless offering
- AWS just runs containers for you based on the CPU / RAM you need



ECR

- Elastic Container Registry
- Private Docker Registry on AWS
- This is where you **store your Docker images** so they can be run by ECS or Fargate



What's serverless?

- Serverless is a new paradigm in which the developers don't have to manage servers anymore…
- They just deploy code
- They just deploy… functions !
- Initially... Serverless == FaaS (Function as a Service)
- Serverless was pioneered by AWS Lambda but now also includes anything that's managed: "databases, messaging, storage, etc."
- **Serverless does not mean there are no servers…**
  it means you just don't manage / provision / see them

So far in this course…

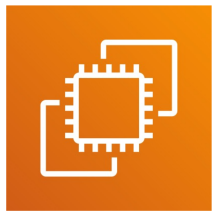
Amazon S3

DynamoDB

Fargate

Lambda

Why AWS Lambda



Amazon EC2

- Virtual Servers in the Cloud
- Limited by RAM and CPU
- Continuously running
- Scaling means intervention to add / remove servers

Amazon Lambda

- Virtual **functions** – no servers to manage!
- Limited by time - **short executions**
- Run **on-demand**
- **Scaling is automated!**

Benefits of AWS Lambda

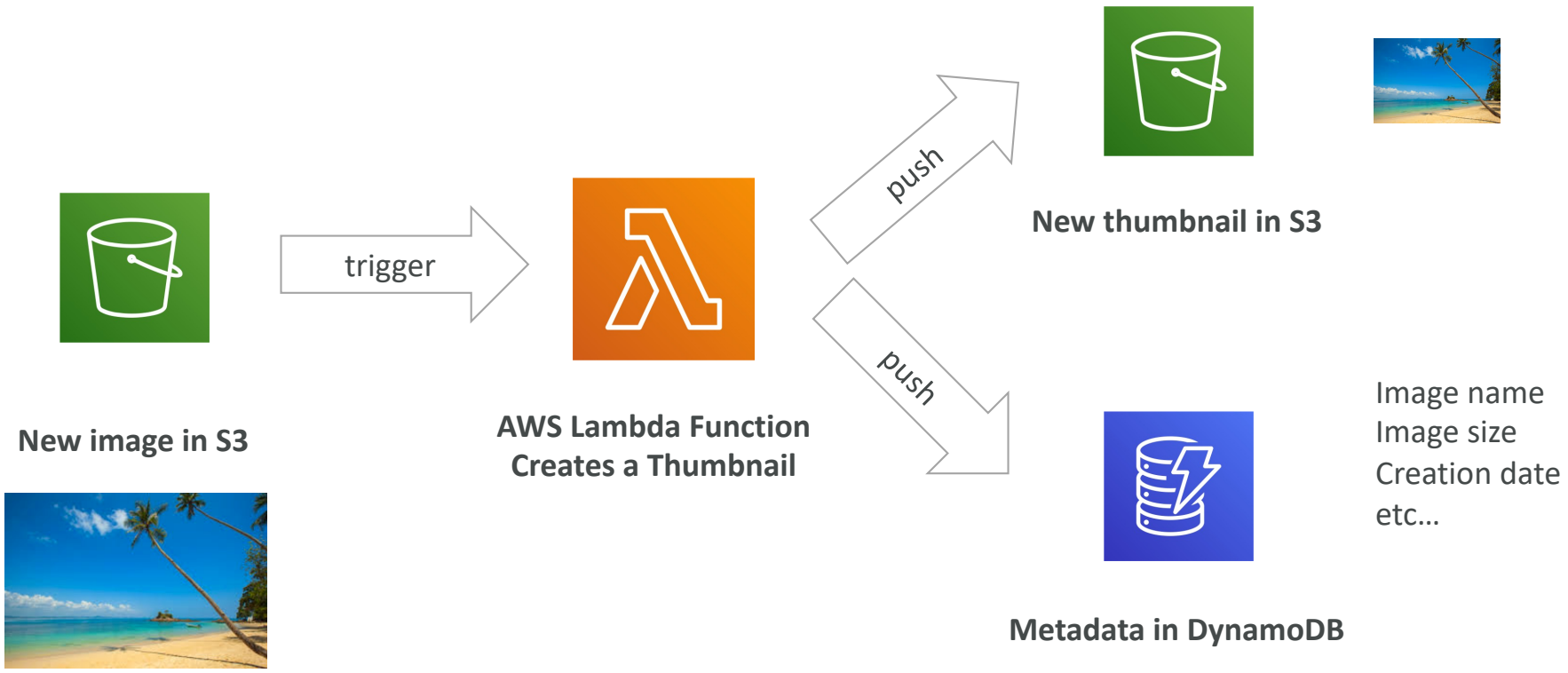
- Easy Pricing:
  - Pay per request and compute time
  - Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time
- Integrated with the whole AWS suite of services
- **Event-Driven:** functions get invoked by AWS when needed
- Integrated with many programming languages
- Easy monitoring through AWS CloudWatch
- Easy to get more resources per functions (up to 10GB of RAM!)
- Increasing RAM will also improve CPU and network!

AWS Lambda language support

- Node.js (JavaScript)
- Python
- Java (Java 8 compatible)
- C# (.NET Core)
- Golang
- C# / Powershell
- Ruby
- Custom Runtime API (community supported, example Rust)

- Lambda Container Image
  - The container image must implement the Lambda Runtime API
  - ECS / Fargate is preferred for running arbitrary Docker images

Example: Serverless Thumbnail creation



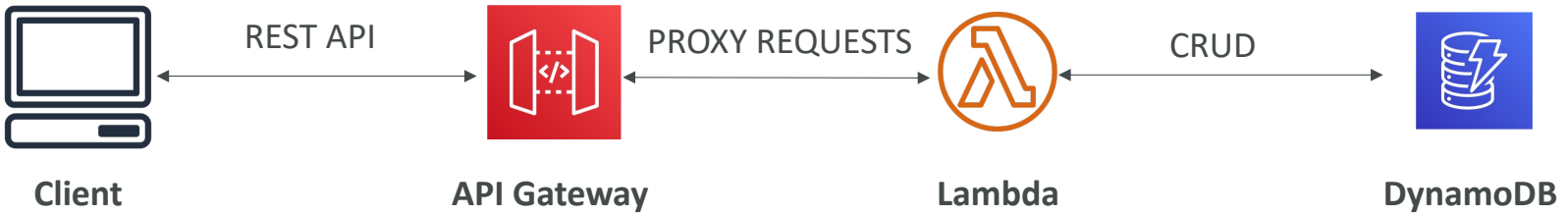
Example: Serverless CRON Job

AWS Lambda Pricing: example

- You can find overall pricing information here: https://aws.amazon.com/lambda/pricing/
- Pay per **calls**:
  - First 1,000,000 requests are free
  - $0.20 per 1 million requests thereafter ($0.0000002 per request)
- Pay per **duration**: (in increment of 1 ms)
  - 400,000 GB-seconds of compute time per month for FREE
  - == 400,000 seconds if function is 1GB RAM
  - == 3,200,000 seconds if function is 128 MB RAM
  - After that $1.00 for 600,000 GB-seconds
- It is usually very cheap to run AWS Lambda so it's very popular

Amazon API Gateway

- Example: building a serverless API

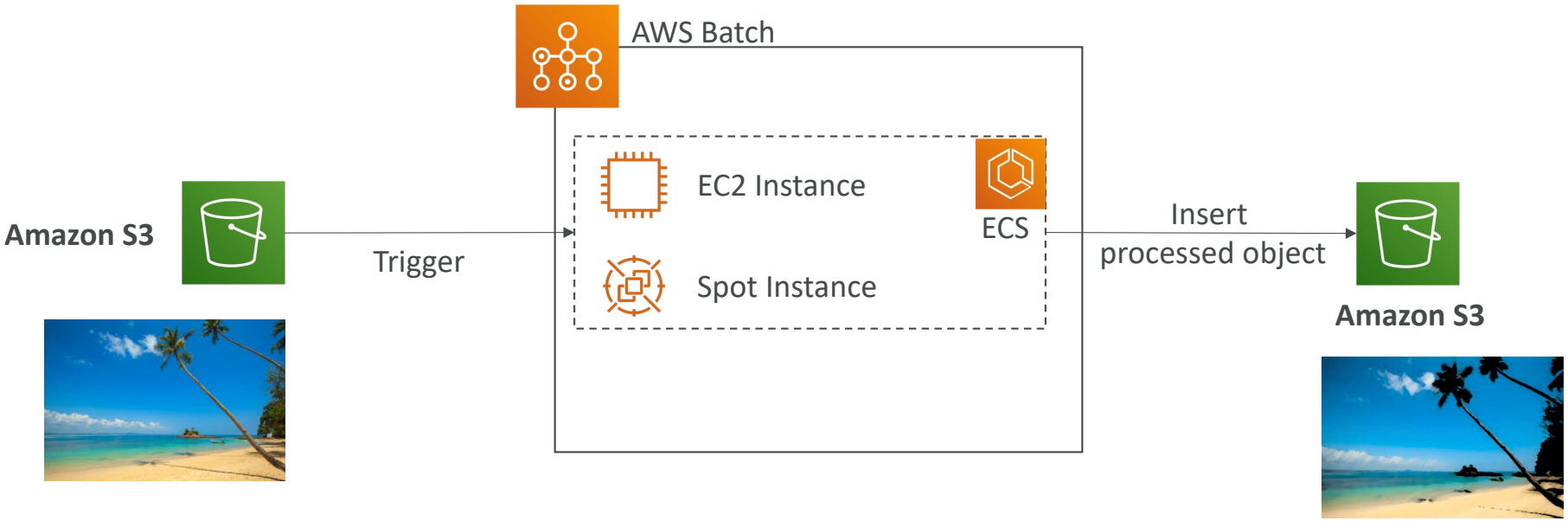


- Fully managed service for developers to easily create, publish, maintain, monitor, and secure APIs
- **Serverless** and scalable
- Supports RESTful APIs and WebSocket APIs
- Support for security, user authentication, API throttling, API keys, monitoring...

AWS Batch

- **Fully managed** batch processing at **any scale**
- Efficiently run 100,000s of computing batch jobs on AWS
- A "batch" job is a job with a start and an end (opposed to continuous)
- Batch will dynamically launch **EC2 instances** or **Spot Instances**
- AWS Batch provisions the right amount of compute / memory
- You submit or schedule batch jobs and AWS Batch does the rest!
- Batch jobs are defined as **Docker images** and **run on ECS**
- Helpful for cost optimizations and focusing less on the infrastructure

AWS Batch – Simplified Example



Batch vs Lambda

- Lambda:
  - Time limit
  - Limited runtimes
  - Limited temporary disk space
  - Serverless

- Batch:
  - No time limit
  - Any runtime as long as it's packaged as a Docker image
  - Rely on EBS / instance store for disk space
  - Relies on EC2 (can be managed by AWS)

Amazon Lightsail

- Virtual servers, storage, databases, and networking
- Low & predictable pricing
- Simpler alternative to using EC2, RDS, ELB, EBS, Route 53…
- Great for people **with little cloud experience**!
- Can setup notifications and monitoring of your Lightsail resources
- Use cases:
  - Simple web applications (has templates for LAMP, Nginx, MEAN, Node.js…)
  - Websites (templates for WordPress, Magento, Plesk, Joomla)
  - Dev / Test environment
- Has high availability but no auto-scaling, limited AWS integrations

Other Compute - Summary

- **Docker:** container technology to run applications
- **ECS:** run Docker containers on EC2 instances
- **Fargate:**
  - Run Docker containers without provisioning the infrastructure
  - Serverless offering (no EC2 instances)
- **ECR:** Private Docker Images Repository
- **Batch:** run batch jobs on AWS across managed EC2 instances
- **Lightsail:** predictable & low pricing for simple application & DB stacks

Lambda Summary

- Lambda is Serverless, Function as a Service, seamless scaling, reactive
- **Lambda Billing:**
  - By the time run x by the RAM provisioned
  - By the number of invocations
- **Language Support:** many programming languages except (arbitrary) Docker
- **Invocation time:** up to 15 minutes
- **Use cases:**
  - Create Thumbnails for images uploaded onto S3
  - Run a Serverless cron job
- **API Gateway**: expose Lambda functions as HTTP API

Deploying and Managing Infrastructure at Scale Section

What is CloudFormation

- CloudFormation is a declarative way of outlining your AWS Infrastructure, for any resources (most of them are supported).
- For example, within a CloudFormation template, you say:
  - I want a security group
  - I want two EC2 instances using this security group
  - I want an S3 bucket
  - I want a load balancer (ELB) in front of these machines
- Then CloudFormation creates those for you, in the **right order**, with the **exact configuration** that you specify

Benefits of AWS CloudFormation (1/2)

- Infrastructure as code
  - No resources are manually created, which is excellent for control
  - Changes to the infrastructure are reviewed through code

- Cost
  - Each resources within the stack is tagged with an identifier so you can easily see how much a stack costs you
  - You can estimate the costs of your resources using the CloudFormation template
  - Savings strategy: In Dev, you could automation deletion of templates at 5 PM and recreated at 8 AM, safely

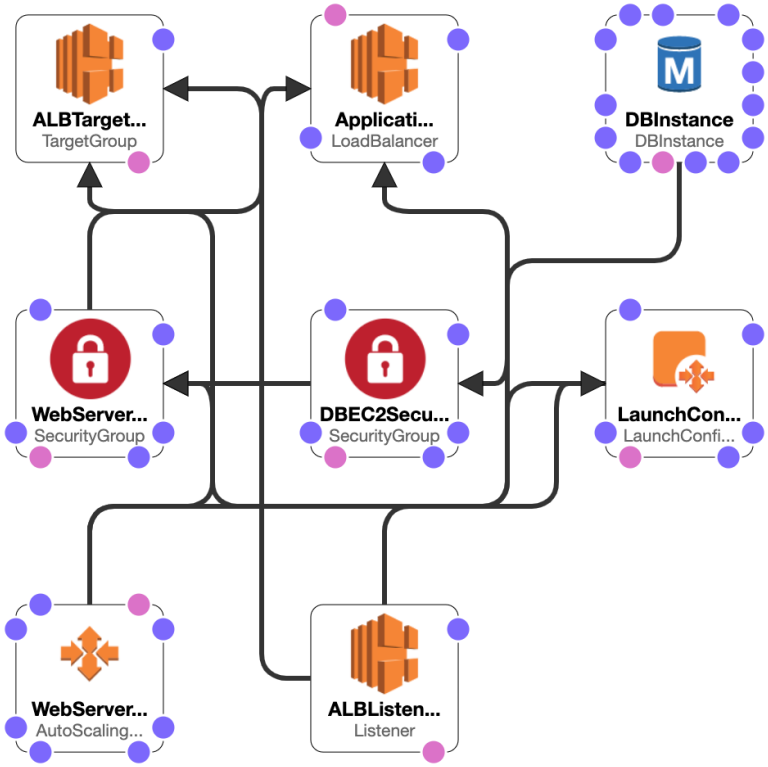
Benefits of AWS CloudFormation (2/2)

- Productivity
  - Ability to destroy and re-create an infrastructure on the cloud on the fly
  - Automated generation of Diagram for your templates!
  - Declarative programming (no need to figure out ordering and orchestration)
- Don't re-invent the wheel
  - Leverage existing templates on the web!
  - Leverage the documentation
- Supports (almost) all AWS resources:
  - Everything we'll see in this course is supported
  - You can use "custom resources" for resources that are not supported

CloudFormation Stack Designer

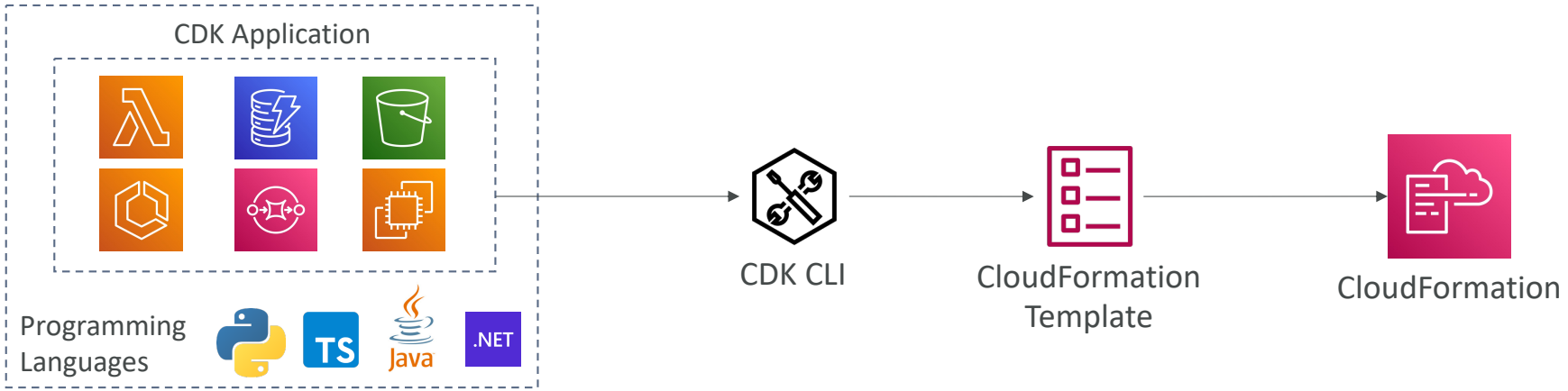
- Example: WordPress CloudFormation Stack

- We can see all the **resources**
- We can see the **relations** between the components



AWS Cloud Development Kit (CDK)

- Define your cloud infrastructure using a familiar language:
  - JavaScript/TypeScript, Python, Java, and .NET
- The code is "compiled" into a CloudFormation template (JSON/YAML)
- You can therefore deploy infrastructure and application runtime code together
  - Great for Lambda functions
  - Great for Docker containers in ECS / EKS



CDK Example

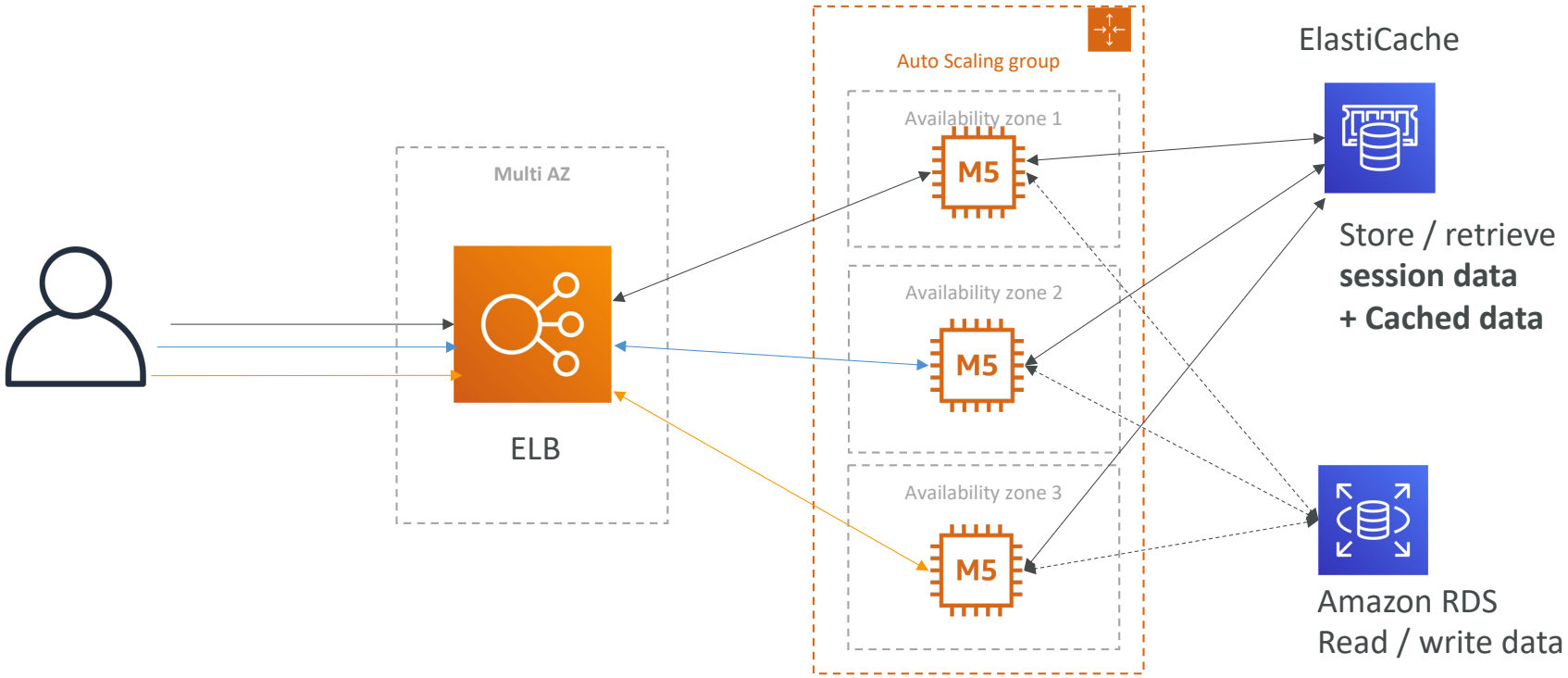
```
export class MyEcsConstructStack extends core.Stack {
  constructor(scope: core.App, id: string, props?: core.StackProps)
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "My
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("am
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is false
    });
  }
}
```

Typical architecture: Web App 3-tier



Developer problems on AWS

- Managing infrastructure
- Deploying Code
- Configuring all the databases, load balancers, etc
- Scaling concerns

- Most web apps have the same architecture (ALB + ASG)
- All the developers want is for their code to run!
- Possibly, consistently across different applications and environments

AWS Elastic Beanstalk Overview

- Elastic Beanstalk is a developer centric view of deploying an application on AWS
- It uses all the component's we've seen before: EC2, ASG, ELB, RDS, etc…
- But it's all in one view that's easy to make sense of!
- We still have full control over the configuration

- **Beanstalk = Platform as a Service (PaaS)**
- Beanstalk is free but you pay for the underlying instances

Elastic Beanstalk

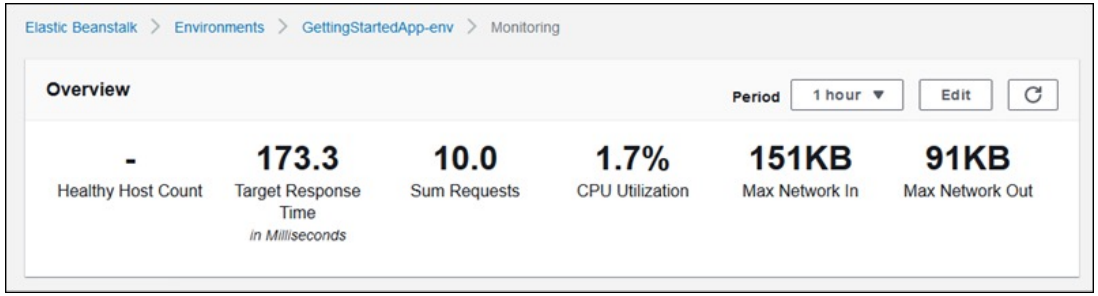
- Managed service
  - Instance configuration / OS is handled by Beanstalk
  - Deployment strategy is configurable but performed by Elastic Beanstalk
  - Capacity provisioning
  - Load balancing & auto-scaling
  - Application health-monitoring & responsiveness
- Just the application code is the responsibility of the developer
- Three architecture models:
  - Single Instance deployment: good for dev
  - LB + ASG: great for production or pre-production web applications
  - ASG only: great for non-web apps in production (workers, etc..)

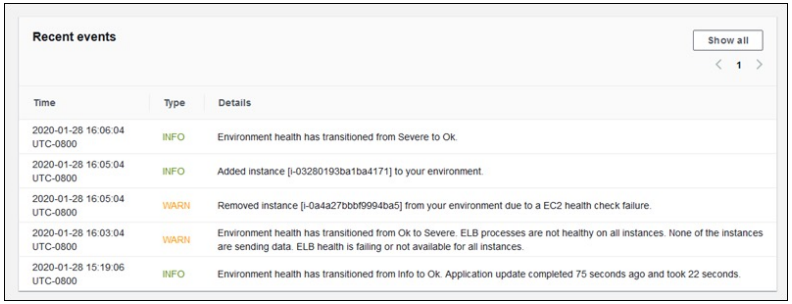
Elastic Beanstalk

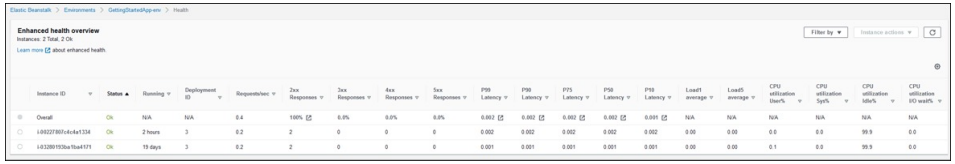
- Support for many platforms:
  - Go
  - Java SE
  - Java with Tomcat
  - .NET on Windows Server with IIS
  - Node.js
  - PHP
  - Python
  - Ruby
  - Packer Builder
  - Single Container Docker
  - Multi-Container Docker
  - Preconfigured Docker
- If not supported, you can write your custom platform (advanced)

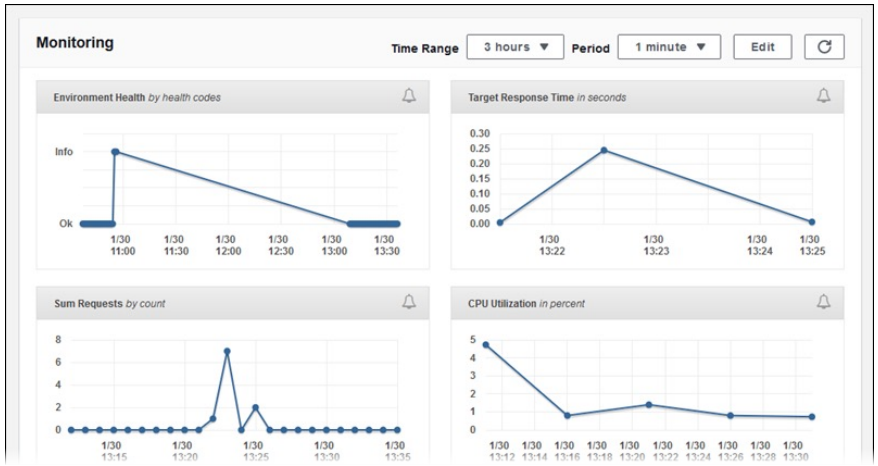
Elastic Beanstalk – Health Monitoring

- Health agent pushes metrics to CloudWatch
- Checks for app health, publishes health events







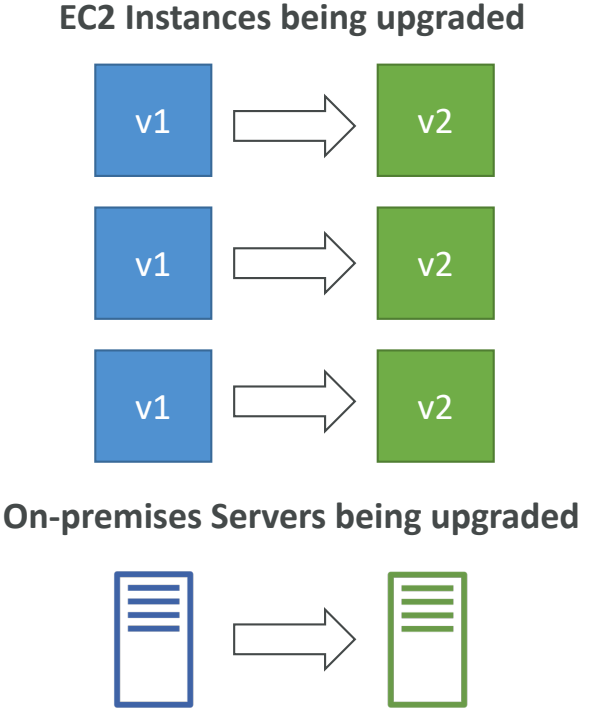


AWS CodeDeploy

- We want to deploy our application **automatically**

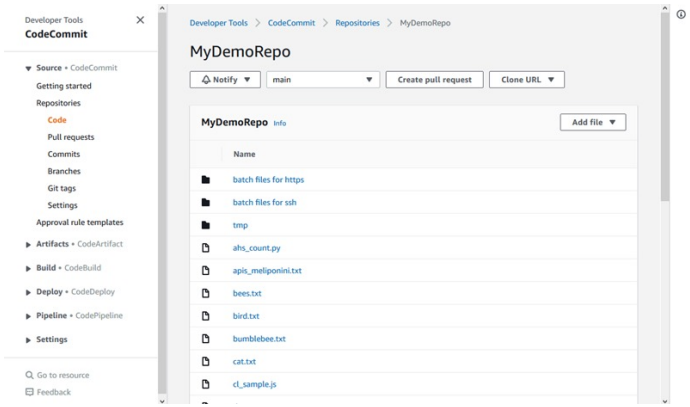
- Works with EC2 Instances
- Works with On-Premises Servers
- Hybrid service

- Servers / Instances must be provisioned and configured ahead of time with the CodeDeploy Agent



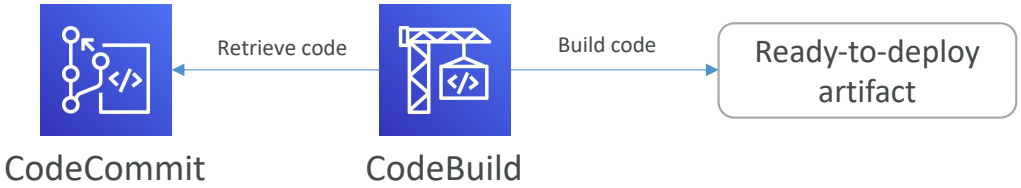
AWS CodeCommit

- Before pushing the application code to servers, it needs to be stored somewhere
- Developers usually store **code in a repository, using the Git technology**
- A famous public offering is GitHub, AWS' competing product is **CodeCommit**
- CodeCommit:
  - Source-control service that **hosts Git-based repositories**
  - Makes it easy to **collaborate with others on code**
  - The code changes are automatically **versioned**
- Benefits:
  - Fully managed
  - Scalable & highly available
  - Private, Secured, Integrated with AWS



AWS CodeBuild

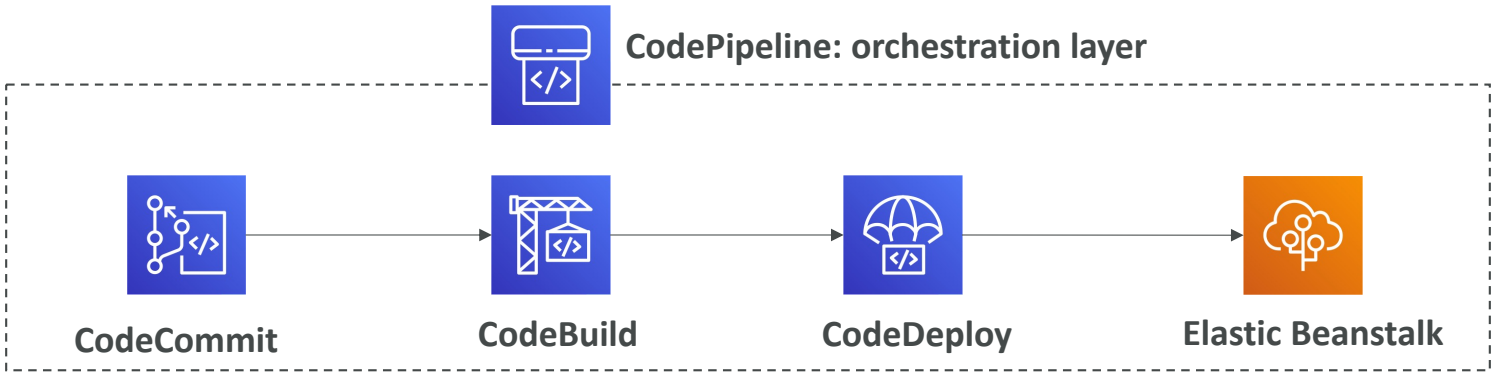
- Code building service in the cloud (name is obvious)
- Compiles source code, run tests, and produces packages that are ready to be deployed (by CodeDeploy for example)



- Benefits:
  - Fully managed, serverless
  - Continuously scalable & highly available
  - Secure
  - Pay-as-you-go pricing – only pay for the build time

AWS CodePipeline

- Orchestrate the different steps to have the code automatically pushed to production
  - Code => Build => Test => Provision => Deploy
  - Basis for CICD (Continuous Integration & Continuous Delivery)
- Benefits:
  - Fully managed, compatible with CodeCommit, CodeBuild, CodeDeploy, Elastic Beanstalk, CloudFormation, GitHub, 3rd-party services (GitHub…) & custom plugins…
  - Fast delivery & rapid updates

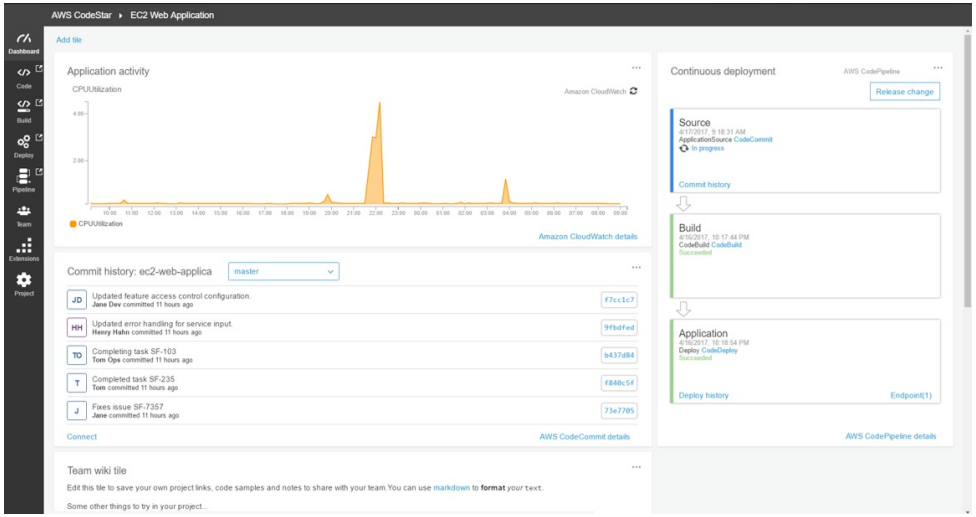


AWS CodeArtifact

- Software packages depend on each other to be built (also called code dependencies), and new ones are created
- Storing and retrieving these dependencies is called **artifact management**
- Traditionally you need to setup your own artifact management system
- **CodeArtifact** is a secure, scalable, and cost-effective **artifact management** for software development
- Works with common dependency management tools such as Maven, Gradle, npm, yarn, twine, pip, and NuGet
- **Developers and CodeBuild can then retrieve dependencies straight from CodeArtifact**

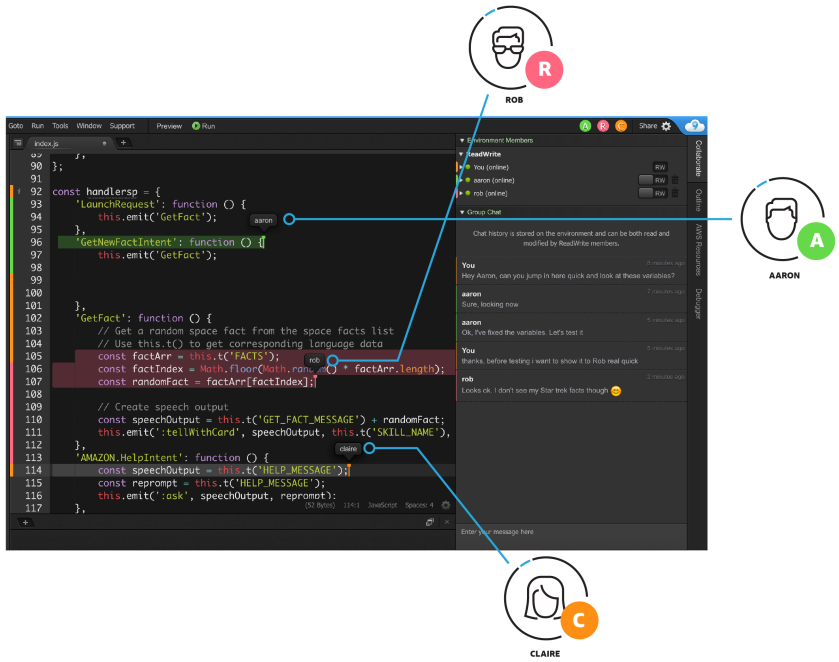
AWS CodeStar

- **Unified UI** to easily manage software development activities **in one place**
- "Quick way" to get started to correctly set-up CodeCommit, CodePipeline, CodeBuild, CodeDeploy, Elastic Beanstalk, EC2, etc…
- Can edit the code "in-the-cloud" using **AWS Cloud9**



AWS Cloud9

- AWS Cloud9 is a cloud IDE (Integrated Development Environment) for writing, running and debugging code
- "Classic" IDE (like IntelliJ, Visual Studio Code…) are downloaded on a computer before being used
- A cloud IDE can be used within a web browser, meaning you can work on your projects from your office, home, or anywhere with internet with no setup necessary
- AWS Cloud9 also allows for code collaboration in real-time (pair programming)

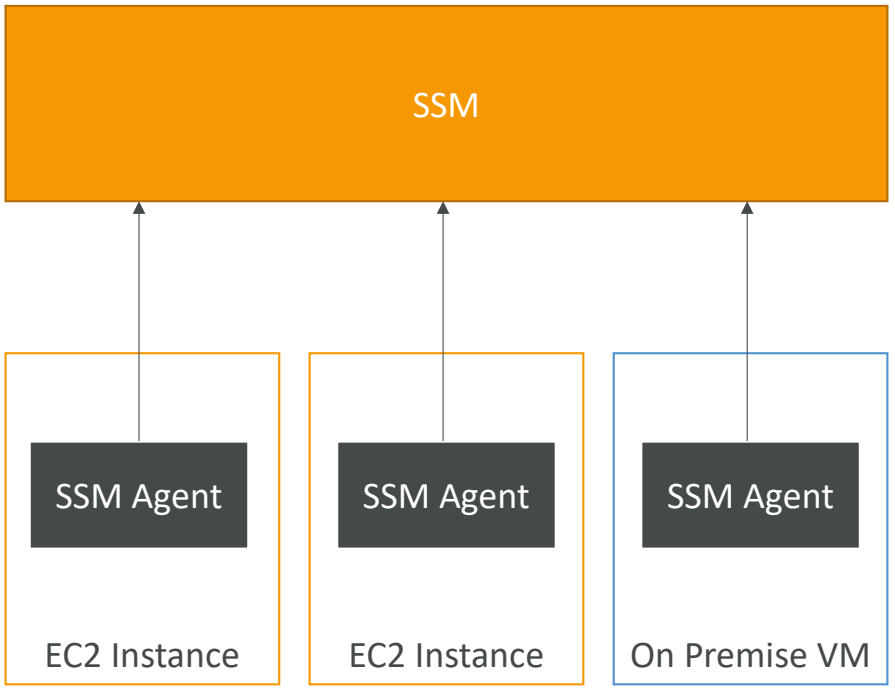


AWS Systems Manager (SSM)

- Helps you manage your **EC2 and On-Premises** systems at scale
- Another **Hybrid** AWS service
- Get operational insights about the state of your infrastructure
- Suite of 10+ products
- Most important features are:
  - **Patching automation for enhanced compliance**
  - **Run commands across an entire fleet of servers**
  - Store parameter configuration with the SSM Parameter Store
- Works for both Windows and Linux OS

How Systems Manager works

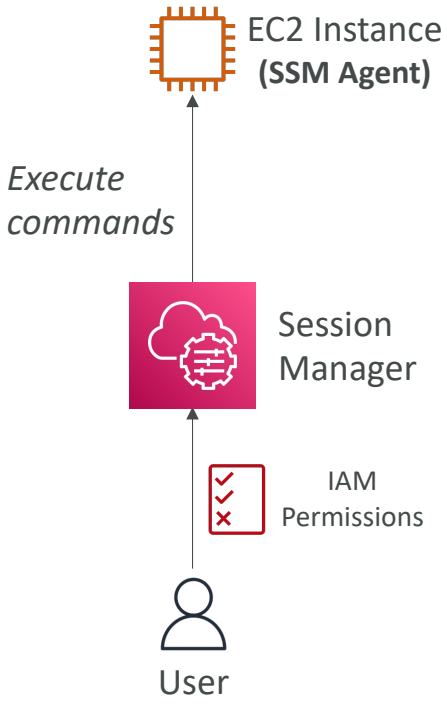
- We need to install the SSM agent onto the systems we control
- Installed by default on Amazon Linux AMI & some Ubuntu AMI
- If an instance can't be controlled with SSM, it's probably an issue with the SSM agent!
- Thanks to the SSM agent, we can **run commands**, **patch** & **configure** our servers



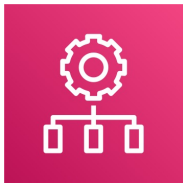
Systems Manager – SSM Session Manager

- Allows you to start a secure shell on your EC2 and on-premises servers

- No SSH access, bastion hosts, or SSH keys needed
- No port 22 needed (better security)
- Supports Linux, macOS, and Windows
- Send session log data to S3 or CloudWatch Logs



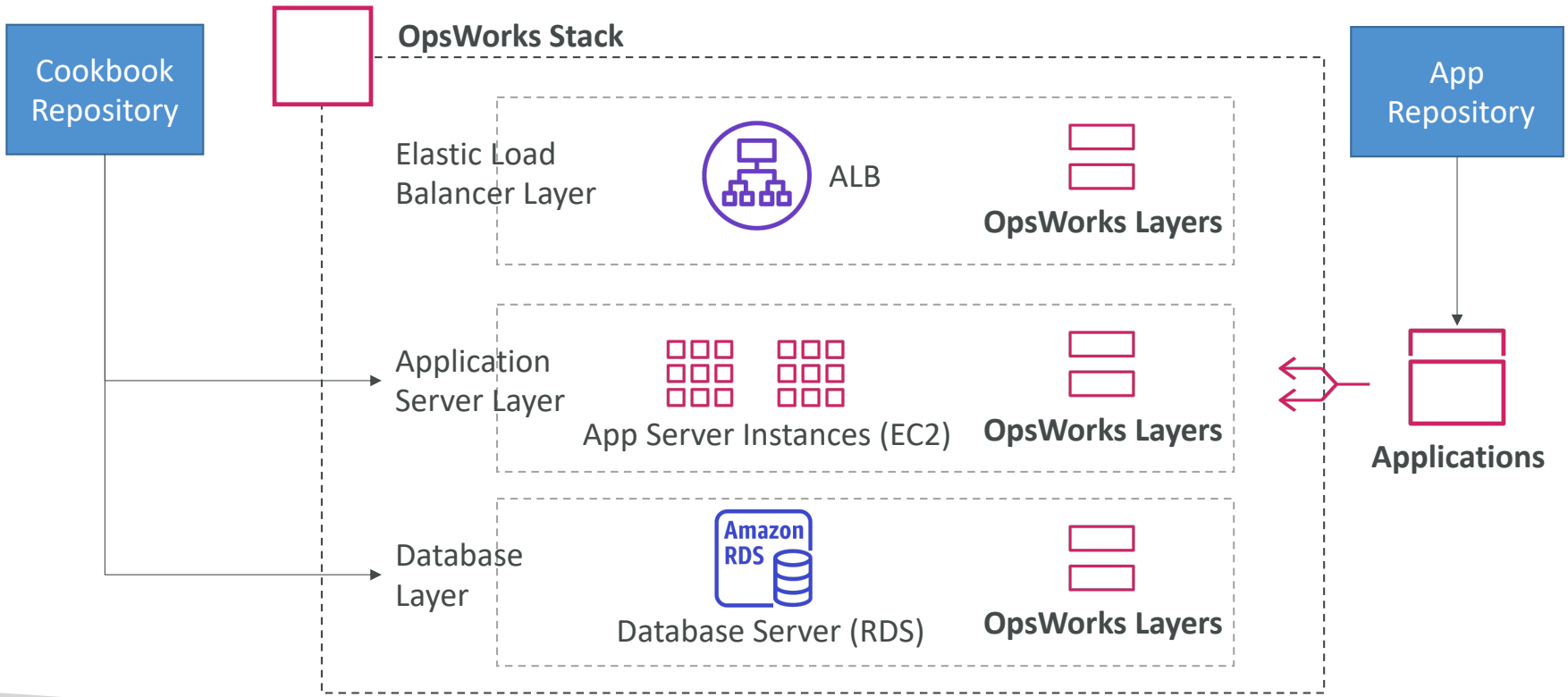
AWS OpsWorks



- Chef & Puppet help you perform server configuration automatically, or repetitive actions
- They work great with EC2 & On-Premises VM
- AWS OpsWorks = Managed Chef & Puppet
- It's an alternative to AWS SSM
- Only provision **standard AWS resources**:
  - EC2 Instances, Databases, Load Balancers, EBS volumes…

- In the exam: Chef or Puppet needed => AWS OpsWorks

OpsWorks Architecture



Deployment - Summary

- **CloudFormation:** (AWS only)
  - Infrastructure as Code, works with almost all of AWS resources
  - Repeat across Regions & Accounts
- **Beanstalk:** (AWS only)
  - Platform as a Service (PaaS), limited to certain programming languages or Docker
  - Deploy code consistently with a known architecture: ex, ALB + EC2 + RDS
- **CodeDeploy** (hybrid): deploy & upgrade any application onto servers
- **Systems Manager** (hybrid): patch, configure and run commands at scale
- **OpsWorks** (hybrid): managed Chef and Puppet in AWS

Developer Services - Summary

- **CodeCommit**: Store code in private git repository (version controlled)
- **CodeBuild**: Build & test code in AWS
- **CodeDeploy**: Deploy code onto servers
- **CodePipeline**: Orchestration of pipeline (from code to build to deploy)
- **CodeArtifact**: Store software packages / dependencies on AWS
- **CodeStar**: Unified view for allowing developers to do CICD and code
- **Cloud9**: Cloud IDE (Integrated Development Environment) with collab
- **AWS CDK**: Define your cloud infrastructure using a programming language