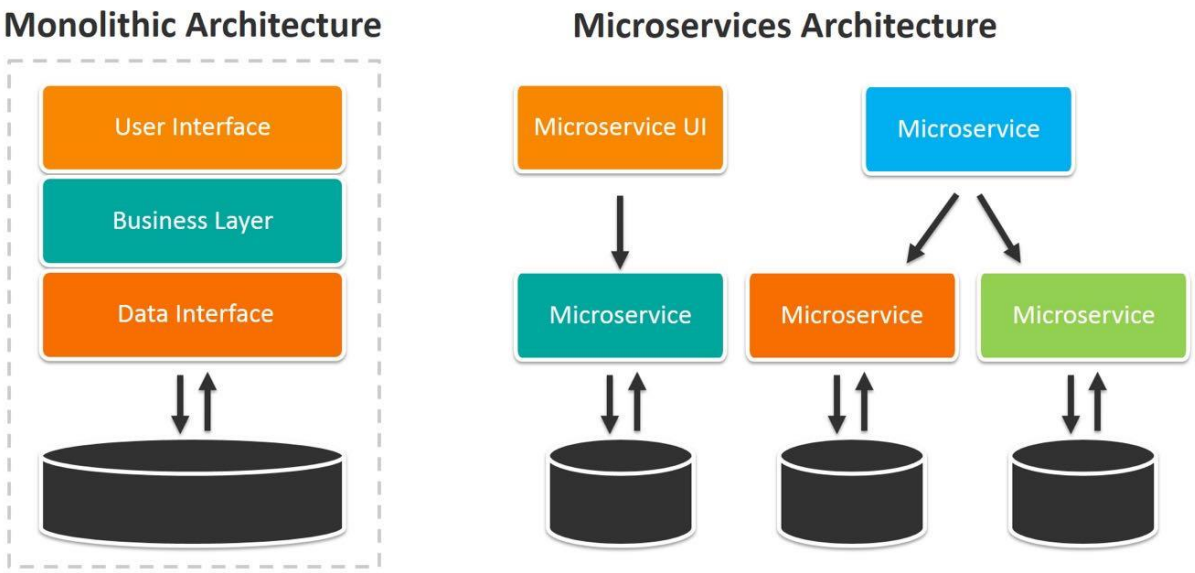
Week 2 - Multi-container deployments, Amazon Elastic Container Service (Amazon ECS) and AWS Fargate.

MICROSERVICE APPLICATIONS

When you design a new application or upgrade an existing application, you make decisions about the architecture of the application. You might have seen a small or older application that has a monolithic architecture. In a monolithic architecture, your application layers are built into a single deployable artifact, and calls between layers are simple function calls. This is a straightforward way to work, but it does have some disadvantages. Each application deployment is a deployment of the entire application, even if you change a single layer. It can also become difficult to mix programming languages if you need to make function calls between languages. A microservice architecture is a popular alternative. With a microservice architecture, your application is decomposed into smaller, independent services. Services run independently and communicate with network protocols. By using independent services, you can do the following:

- Scale and deploy individual services
- Build teams around services that match business functions
- Choose the correct programming language and technology for each service

The rapid pace of deployment in a distributed microservices architecture lends itself to containerization. If you need to make a change to a single service, you can deploy containers of the new version of the service and switch all communications to the new containers.



DOCKER COMPOSE

Docker Compose is a tool for automating tasks in a multi-container environment. You can define the container settings—such as ports, volumes, and commands—into a single YAML file. You can use a single Docker Compose command to launch multiple containers from the YAML file. Starting with a sample docker-compose.yaml file, you can see how Docker parameters are defined into a declarative file.

services: wordpress: image: wordpress:latest ports: - "80:80" database: image: mysql:5.7 environment: MYSQL_RANDOM_ROOT_PASSWORD: 1 MYSQL_DATABASE: wordpress

The example YAML file creates two containers. The image setting sets the Docker image that will be used for the container. The ports setting defines the container ports that are made available on the host. The database container is configured with environment variables in the environment settings. The simple example can be the following command: docker compose up. Docker Compose launches containers with the provided parameters, and creates a Docker network for the containers to communicate. These are all tasks you could perform by using Docker commands. Docker Compose can automate these steps for you. After the containers launch, you can use some additional Docker Compose commands to interact with the containers and inspect them, such as the following:

- docker compose exec: Run commands inside of the containers (for example, launch a shell inside the container environment)
- docker compose logs: Inspect log output
- docker compose down: Stop and remove the running containers

For more information, see the Overview of Docker Compose.

Comparing the control plane and the data plane

The *control plane* consists of the cluster control components that expose APIs and interfaces to define, deploy, and manage the lifecycle of containers. For Amazon Elastic Container Service (Amazon ECS), you interact with the control plane by using the Amazon ECS API. The *data plane* is the infrastructure that provides the capacity for containers to run, such as CPU, memory, network, and storage.

Amazon ECS

Amazon ECS is a highly scalable and **fast container-management service**. You can use it to **run, stop, and manage containers on a cluster.**

With Amazon ECS, your containers are defined in a task definition that you use to run an individual task or a task within a service. In this context, a *service* is a configuration that you can use to run and maintain a specified number of tasks simultaneously in a cluster. You can run your tasks and services on a serverless infrastructure that's managed by AWS Fargate. Alternatively, for more control over your infrastructure, you can run your tasks and services on a cluster of Amazon Elastic Compute Cloud (Amazon EC2) instances that you manage.For more information about Amazon ECS, see What is Amazon Elastic Container Service?

Core concepts for Amazon ECS

- Cluster: A logical grouping of tasks or services. Your tasks and services are run on infrastructure that’s registered to a cluster, such as AWS Fargate nodes, EC2 instances, or remote virtual machines (VMs). For more information about clusters for Amazon ECS, see Amazon ECS clusters.
- Container instances: A container instance is an EC2 instance that’s a part of your Amazon ECS cluster. A container instance isn’t the same as a task, which contains a running instance of your container. For more information about container instances, see Amazon ECS container instances.
- Container agent: The container agent allows container instances to connect to your Amazon ECS cluster. For more information about the Amazon ECS container agent, see Amazon ECS container agent.
- Task definition: A task definition contains configuration information for the containers you want to run on Amazon ECS. You need to create a task definition to run containers on Amazon ECS. Task definitions include parameters, such as the following:
    - The Docker image to use with each container in your task.
    - How much CPU and memory to use with each task or each container within a task.

- The launch type to use, which determines the infrastructure where your tasks are hosted.
- The Docker networking mode to use for the containers in your task.
- The logging configuration to use for your tasks.
- Whether the task should continue to run if the container finishes or fails.
- The command the container should run when it's started.
- Any data volumes that should be used with the containers in the task.
- The AWS Identity and Access Management (IAM) role that your tasks should use.

For more information about task definitions, see Amazon ECS task definitions.

- Task: A task is a running instantiation of your task definition.
- Service: A service runs a specified number of tasks simultaneously on your cluster. For more information about services, see Amazon ECS services.

Scheduling tasks with Amazon ECS

To run a task on Amazon Elastic Container Service (Amazon ECS), you must schedule it. The scheduling engine determines how and when to launch a task. There are multiple ways you can schedule tasks: through service scheduling, manual scheduling, cron scheduling, or by using custom schedulers. For more information about scheduling tasks on Amazon ECS, see Scheduling Amazon ECS tasks.

Placing tasks with Amazon ECS

After you determine when you want to run a task through scheduling, Amazon ECS then needs to determine where to run the task, and how to place that task based on requirements that you define. For more information about task placement on Amazon ECS, see Amazon ECS task placement.

Launch types

For Amazon ECS, the launch type you choose, either EC2 or Fargate, is chosen when you run a service or launch a task.

- Fargate launch type: Run your containerized applications without the need to provision and manage the backend infrastructure. AWS Fargate is the serverless way to host your Amazon ECS workloads.

- EC2 launch type: Run your containerized applications on Amazon EC2 instances that you register to your Amazon ECS cluster and manage yourself.

For more information about launch types, see Amazon ECS Launch types.

Capacity providers

Capacity providers manage the infrastructure that the tasks in your clusters use. For more information about capacity providers and how to use them, see this blog post by Adam Keller: Managing compute for Amazon ECS clusters with capacity providers.For more detailed information about capacity providers, see Amazon ECS capacity providers.

Scaling your cluster

Amazon ECS cluster auto scaling provides control over how you scale the Amazon Elastic Compute Cloud (Amazon EC2) instances within a cluster. When you use managed scaling, Amazon ECS creates the infrastructure for the Auto Scaling group capacity provider, and manages the scale-in and scale-out actions of the Auto Scaling group based on your clusters' task load. For more information about cluster auto scaling, see Amazon ECS cluster auto scaling.

Scaling your tasks

To scale your tasks automatically based on demand, you can use service auto scaling. Amazon ECS publishes Amazon CloudWatch metrics with your service's average CPU and memory usage. You can then use these metrics to scale your service as demand changes. For more information about scaling tasks, see Service auto scaling.

Service Discovery with AWS Cloud Map

Amazon ECS uses AWS Cloud Map for service discovery. For more information about how Amazon ECS integrates with AWS Cloud Map, see Service Discovery.

AWS Fargate

AWS Fargate is a serverless compute platform for your Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Kubernetes Service (Amazon EKS) clusters. You can use Fargate instead of deploying your Amazon ECS tasks and Amazon EKS Pods to Amazon Elastic Compute Cloud (Amazon EC2) instances in your account. With Fargate, you don't need to maintain and deploy servers. In addition, you pay only for the resources you use. Your Pods and tasks are run in an isolated runtime environment for improved security. For more information, see the AWS Fargate FAQ.

AWS Copilot

AWS Copilot is a command line tool that you can use with Amazon Elastic Container Service (Amazon ECS). Copilot automates the creation of many resources that you need to host your container workloads on Amazon ECS. To deploy a simple website with AWS Copilot, you need to create three things: an application, an environment, and a service. An application is a logical grouping of resources. An environment is used for different settings and stages of your application (such as testing, staging, and production). Your application can have multiple services, which are where your code lives. Services can be one of several different types. The example application in this course creates two services: a *load balanced web service* for the frontend, and a *backend service* for the API. Each service uses your Dockerfile to define an application container image. Copilot creates Amazon Elastic Container Registry (Amazon ECR) repositories. It then builds and pushes the container images as part of the automated deployment. Copilot creates the Amazon ECS and load balancer resources that are needed to run your service inside Amazon ECS. These resources are different for each service type. For more information, see Services in the AWS Copilot CLI documentation. Everything created by Copilot is defined in a manifest file. The manifest file contains all the details that are used to deploy your services, such as scaling the number of tasks hosting your application. Source-controlling your manifest file is a good way to follow some DevOps practices. When the state of your deployed applications is committed to source control, where you have a full history of all changes. By using source control, you can capture much of your team's knowledge about your applications. If you created Amazon ECS application resources manually, you could lose this knowledge and deployments might no longer be repeatable. For more information about Copilot, see the AWS Copilot CLI documentation.