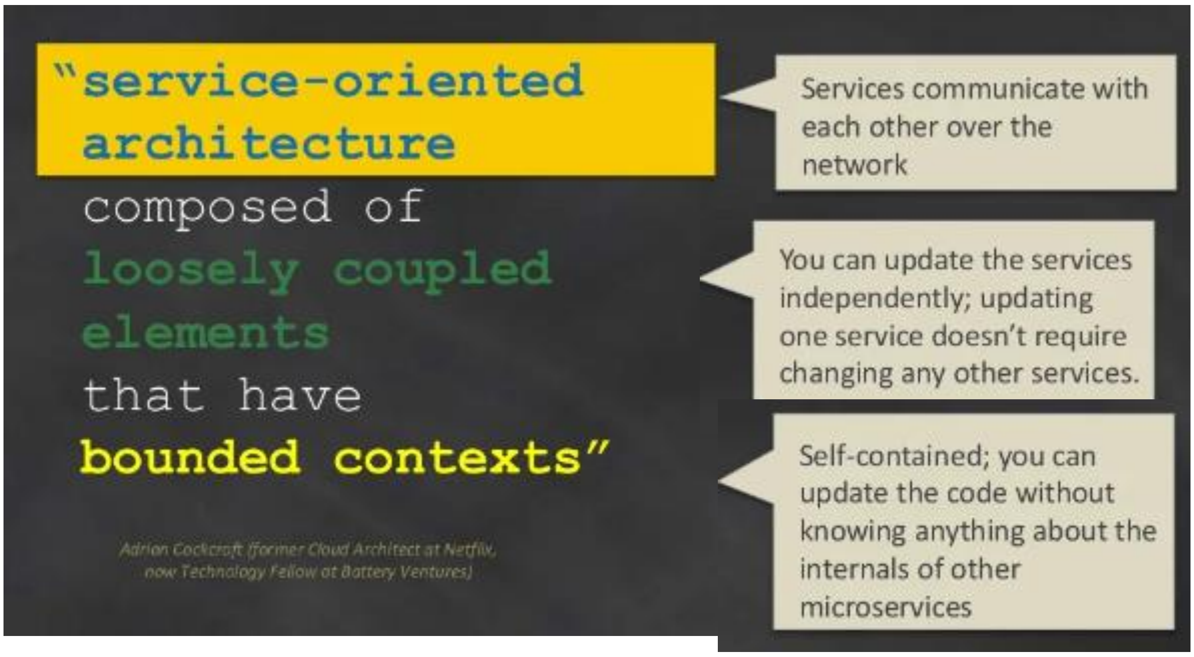
1.Microservices



"service-oriented architecture composed of loosely coupled elements that have bounded contexts"

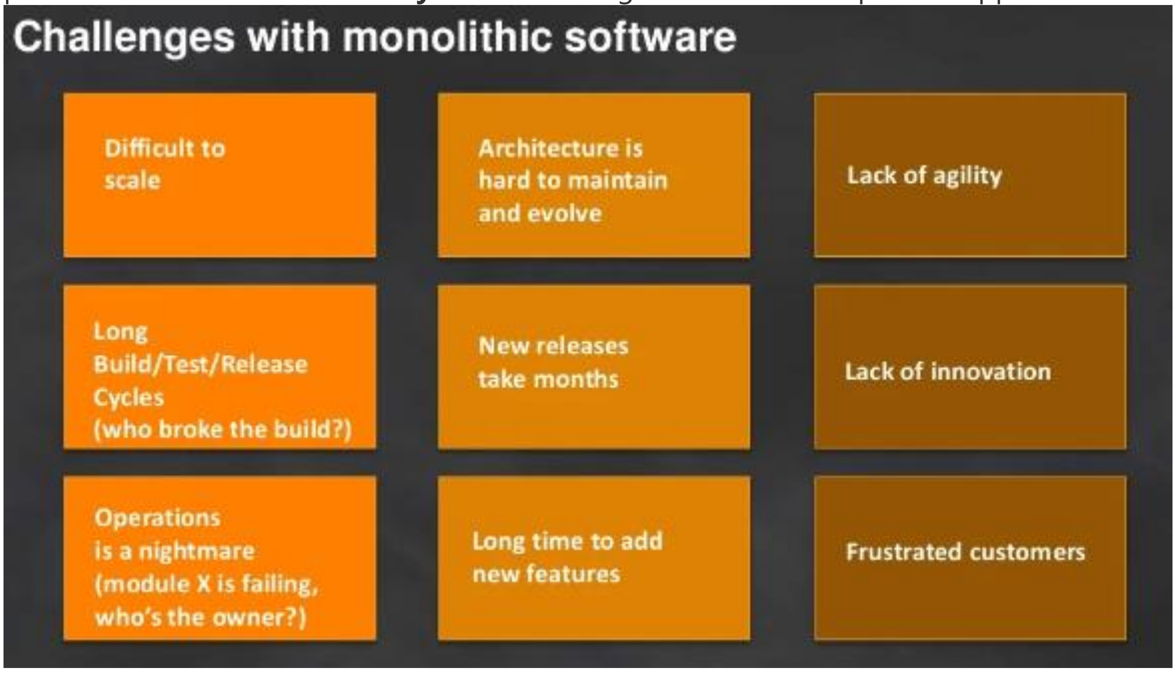
Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

Services communicate with each other over the network

You can update the services independently; updating one service doesn't require changing any other services.

Self-contained; you can update the code without knowing anything about the internals of other microservices

- These are the services which are exposed by REST.
- These are small well-chosen deployable units.
- The most important feature of the microservice-based architecture is that it can perform **continuous delivery** of a large and complex application.



Challenges with monolithic software

Difficult to scale	Architecture is hard to maintain and evolve	Lack of agility
Long Build/Test/Release Cycles (who broke the build?)	New releases take months	Lack of innovation
Operations is a nightmare (module X is failing, who's the owner?)	Long time to add new features	Frustrated customers

- API gateway

REST API - REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways.

When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation)

REST template

@Bean is just for the metadata definition to create the bean(equivalent to tag).

@Autowired is to inject the dependancy into a bean(equivalent to ref XML tag/attribute).

2.Docker

Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels.

Important Terminologies in Docker

1. Docker Image
- It is a file, comprised of multiple layers, used to execute code in a Docker container.
- They are a set of instructions used to create docker containers.

2. Docker Container
- It is a runtime instance of an image.
- Allows developers to package applications with all parts needed such as libraries and other dependencies.

3. Docker file
- It is a text document that contains necessary commands which on execution helps assemble a Docker Image.
- Docker image is created using a Docker file.

4. Docker Engine
- The software that hosts the containers is named Docker Engine.
- Docker Engine is a client-server based application
- The docker engine has **3 main** components:
    - **Server**: It is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.
    - **REST API**: It specifies how the applications can interact with the Server and instructs it what to do.

- **Client**: The Client is a docker command-line interface (CLI), that allows us to interact with Docker using the docker commands.

5. Docker Hub

- Docker Hub is the official online repository where you can find other Docker Images that are available for use.
- It makes it easy to find, manage, and share container images with others.

3.Orchestrator

Orchestrator **manages the creation, monitoring, scheduling, and controlling of automated bots and processes**. It's a centralized forum for managing and controlling all software bots

Kubernetes is a popular open source platform for container orchestration. It enables developers to easily build containerized applications and services, as well as scale, schedule and monitor those containers

Architecture of Kubernetes

Kubernetes follows the client-server architecture where we have master installed on one machine and the node on separate Linux machines. It follows the masterslave model, which uses a master to manage Docker containers across multiple Kubernetes nodes. A master and its controlled nodes(worker nodes) constitute a **"Kubernetes cluster"**. A developer can deploy an application in the docker containers via the assistance of the Kubernetes master.

1. Kubernetes-Master Node Components –

Kubernetes master is responsible for managing the entire cluster, coordinates all activities inside the cluster and communicates with the worker nodes to keep the Kubernetes and your application running. This is the entry point of all administrative tasks. When we install Kubernetes on our system we have four primary components of Kubernetes Master that will get installed. The components of Kubernetes Master node are:

a.) API Server– The API server is the entry point for all the REST commands used to control the cluster. All the administrative tasks are done by API server within the master node. If we want to create, delete, update or display in Kubernetes object it has to go through this API server.API server validates and configures the API objects such as ports, services, replication, controllers and deployments and it is responsible for exposing API's for every operation. We can interact with this API's using a tool called **kubcetl**. *'kubcetl' is a very tiny go language binary which basically talks to the API server to perform any operations that we issue from the command line. It is a command-line interface for running commands against Kubernetes clusters*

b.) Scheduler– It is a service in master responsible for distributing the workload. It is responsible for tracking the utilization of working load of each worker nodes and then placing the workload on which resources are available and can accept the workload. The scheduler is responsible for

scheduling pods across available nodes depending on the constraints you mention in the configuration file it schedule these pods accordingly. The scheduler is responsible for workload utilization and allocating pod to new node.
c.) Controller Manager– Also known as controllers.It is a daemon which runs in nonterminating loop and is responsible for collecting and sending information to API server. It regulates the kubernetes cluster by performing lifestyle function such as namespace creation and lifecycle event garbage collections, terminated pod garbage collection, cascading deleted garbage collection, node garbage collection and many more. Basically controller watches the desired state of cluster if the current state of the cluster does not meet the desired state then the control loop takes the corrective steps to make sure that the current state is same as that of the desired state. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. So in this way controllers are responsible for overall health of the entire cluster by ensuring that nodes are up and running all the time and correct pods are running as mentioned in the specs file.
d.) etcd– It is a distributed key-value lightweight database. In Kubernetes, it is a central database for storing the current cluster state at any point of time and also used to store the configuration details such as subnets, config maps etc. It is written in Go programming language.

2. Kubernetes-Worker Node Components

a.) Kubelet– It is a primary node agent which communicates with the master node and executes on each worker node inside the cluster. It gets the pod specifications through the API server and execute the container associated with the pods and ensures that the containers described in the pods are running and healthy. If kubelet notices any issues with the pods running on the worker nodes then it tries to restart the pod on the same node and if the issue is with the worker node itself then the Kubernetes master node detects the node failure and decides to recreate the pods on the other healthy node.
b.) Kube-Proxy– It is the core networking component inside the kubernetes cluster. It is responsible for maintaining the entire network configuration. Kube-Proxy maintains the distributed network across all the nodes, all the pods and across all the containers and also exposes the services across the outside world. It acts as a network proxy and load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets. It listens to the API server for each service endpoint creation and deletion so for each service endpoint it sets up the route so that you can reach to it.
c.) Pods– Pod is a group of containers that are deployed together on the same host. With the help of pods we can deploy multiple dependent containers together so it acts as a wrapper around these containers so we can interact and manage these containers primarily through pods.

Sidecar

- The sidecar containers can also share storage volumes with the main containers, allowing the main containers to access the data in the sidecars.
- The main and sidecar containers also share the pod network, and the pod containers can communicate with each other on the same network using localhost or the pod’s IP, reducing latency between them.