Cloud & CI/CD

Jonas Blum, Thomas Fritz

Agenda

1. Software in the cloud
2. Continuous Integration, Delivery & Deployment
3. Software Release Planning

Examinable Skills

By the end of this lecture, you should be able to…

- explain characteristics as well as pros/cons of the cloud, the different cloud layers and cloud development
- discuss the move of an application to the cloud and what it entails
- explain the differences, benefits and importance of continuous * (integration, delivery, deployment)
- explain the importance of release planning
- explain, apply and justify principles and patterns for release management and deployment

Why cloud?

Home / Innovation / Cloud

Roche moving over 90,000 employees to Google Apps

Like many other businesses making the switch to Google Apps and other cloud-based solutions, The Roche Group suffered from "platform interoperability issues."

How Lufthansa is flying its data warehouse to the cloud

Moving from an on-premises data system to the cloud

The Economic Times

Interview | AWS CEO on $12.7 billion India investment, ChatGPT…

1 day ago

Supply Chain Digital

Volkswagen Group selects AWS to build Volkswagen Industrial Cloud platform

This week, auto manufacturer Volkswagen Group announced that it has selected Amazon Web Services for a multi-year global agreement to build...

17 May 2020

NETFLIX

← Back to All News

Completing the Netflix Cloud Migration

Netflix, 2016

PAAS + IAAS

10

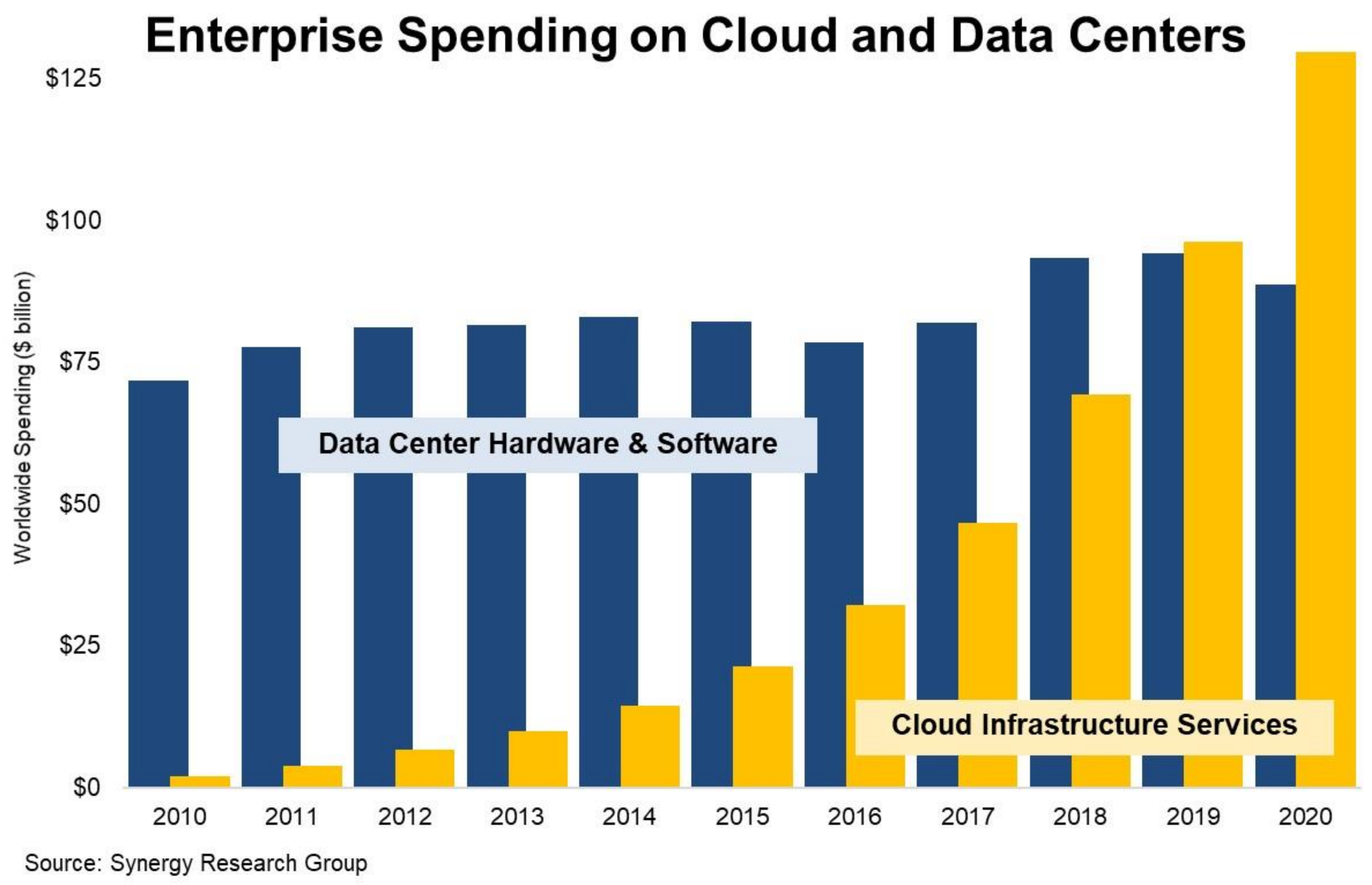
UBS throws lot in with Microsoft, migrating 50% of apps to Azure

Five-year blueprint to modernize tech estate, but cloud still not for everyone in finance sector

Paul Kunert

Thu 20 Oct 2022 // 17:15 UTC

Why cloud?

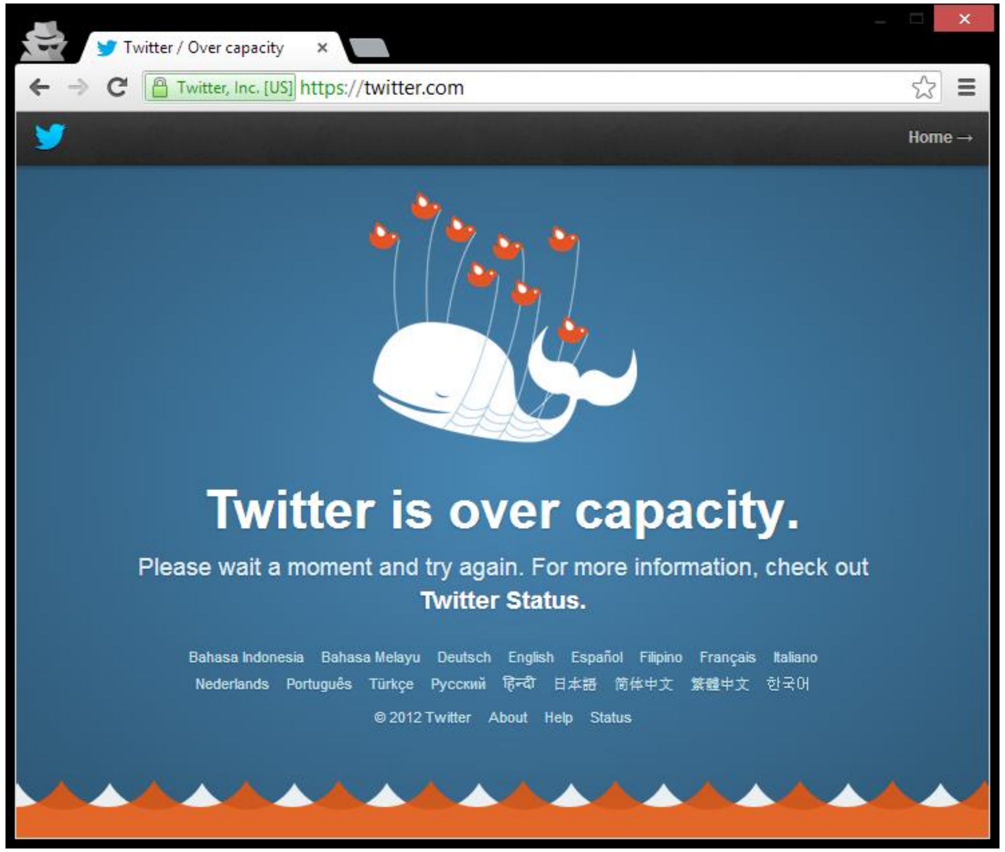


https://www.srgresearch.com/articles/2020-the-year-that-cloud-service-revenues-finally-dwarfed-enterprise-spending-on-data-centers

The cloud seems to be important

What issues does the cloud solve?

Why cloud?



Twitter / Over capacity

Twitter, Inc. [US] https://twitter.com

Home →

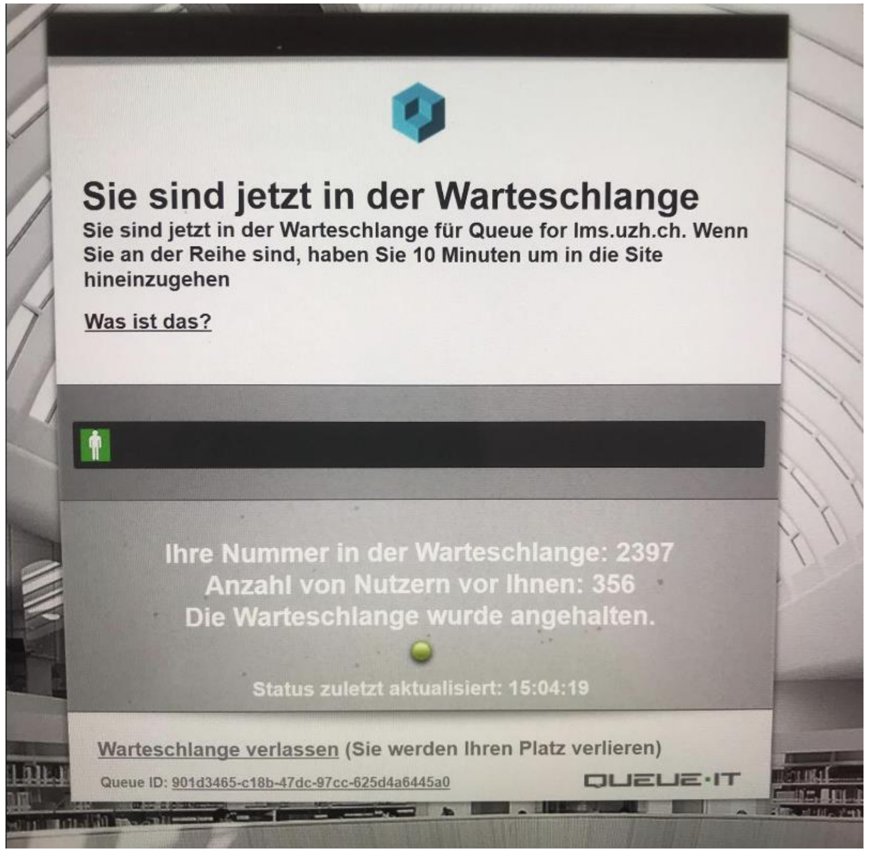
Twitter is over capacity.

Please wait a moment and try again. For more information, check out **Twitter Status.**

Bahasa Indonesia Bahasa Melayu Deutsch English Español Filipino Français Italiano Nederlands Português Türkçe Русский हिन्दी 日本語 简体中文 繁體中文 한국어

© 2012 Twitter About Help Status

Twitter's Fail Whale



Sie sind jetzt in der Warteschlange

Sie sind jetzt in der Warteschlange für Queue for lms.uzh.ch. Wenn Sie an der Reihe sind, haben Sie 10 Minuten um in die Site hineinzugehen

Was ist das?

Ihre Nummer in der Warteschlange: 2397
Anzahl von Nutzern vor Ihnen: 356
Die Warteschlange wurde angehalten.

Status zuletzt aktualisiert: 15:04:19

Warteschlange verlassen (Sie werden Ihren Platz verlieren)

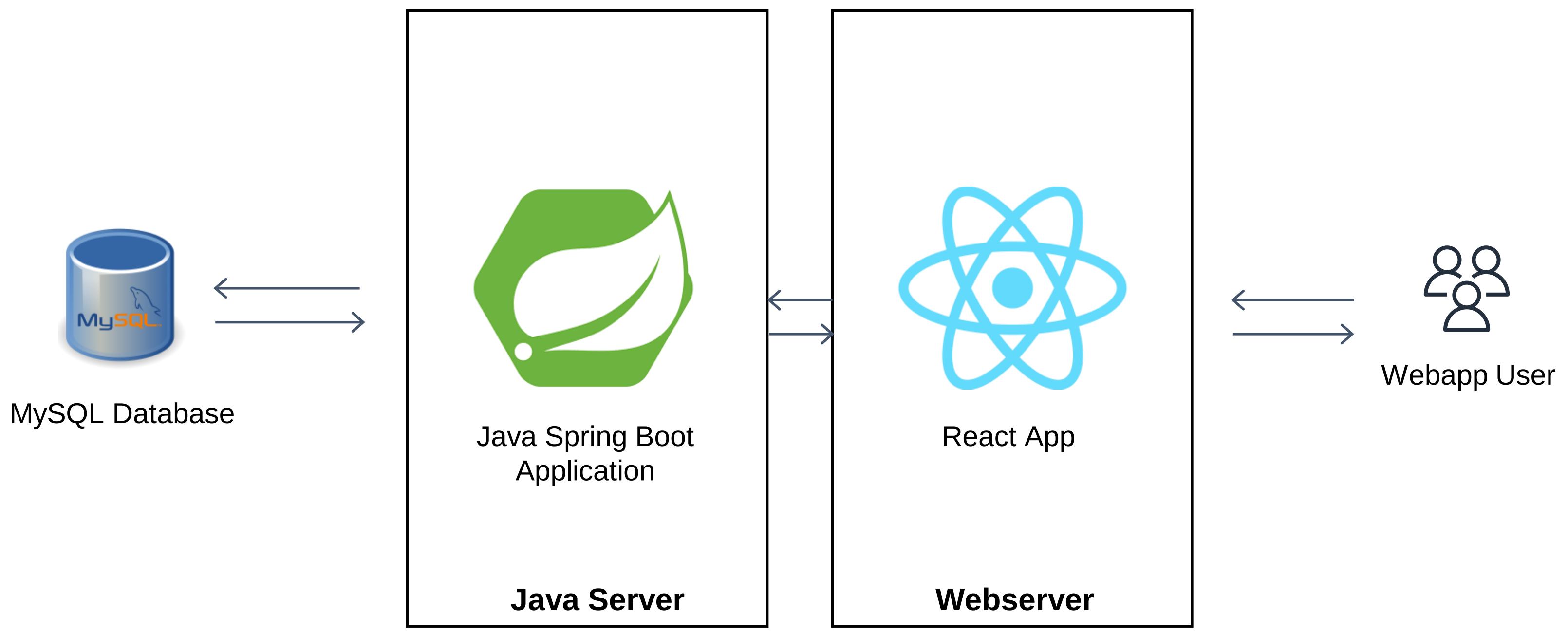
Queue ID: 901d3465-c18b-47dc-97cc-625d4a6445a0

QUEUE-IT

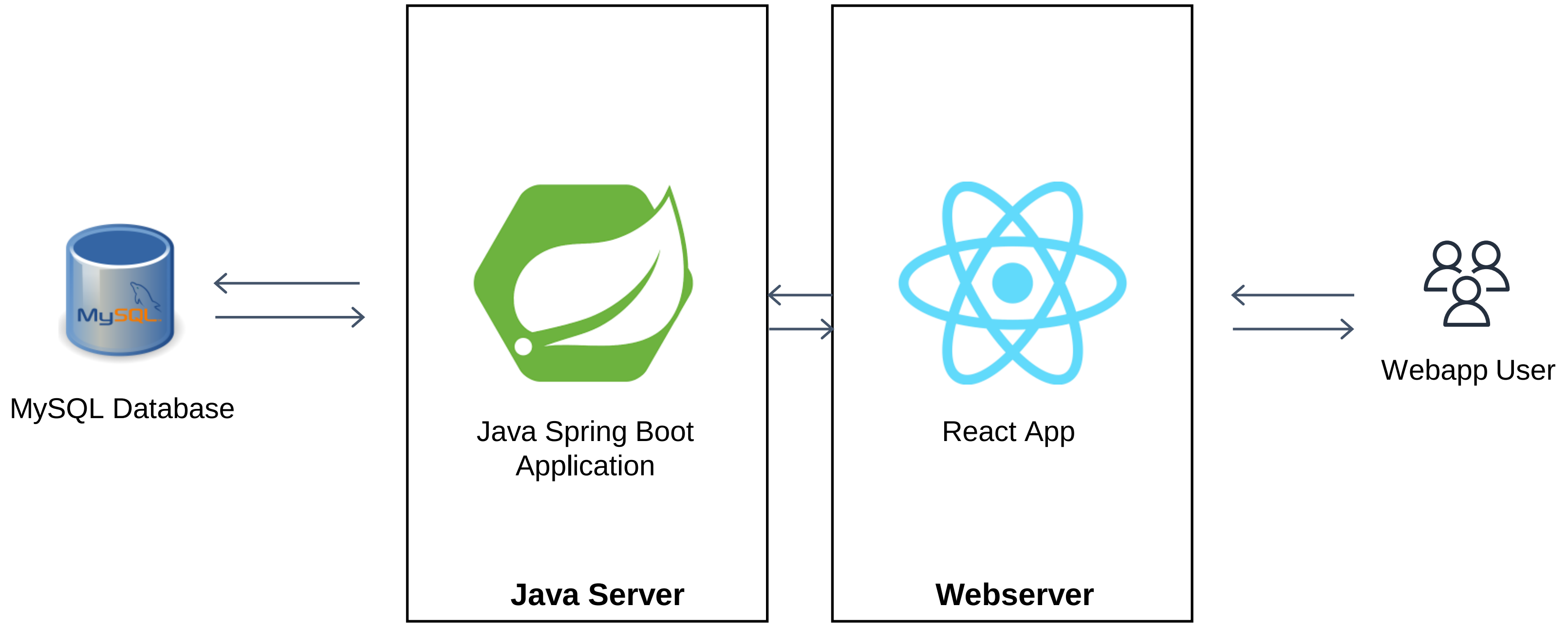
The OLAT Queue (during Covid)

Why should you care about the cloud?

Traditional Backend-Frontend Architecture



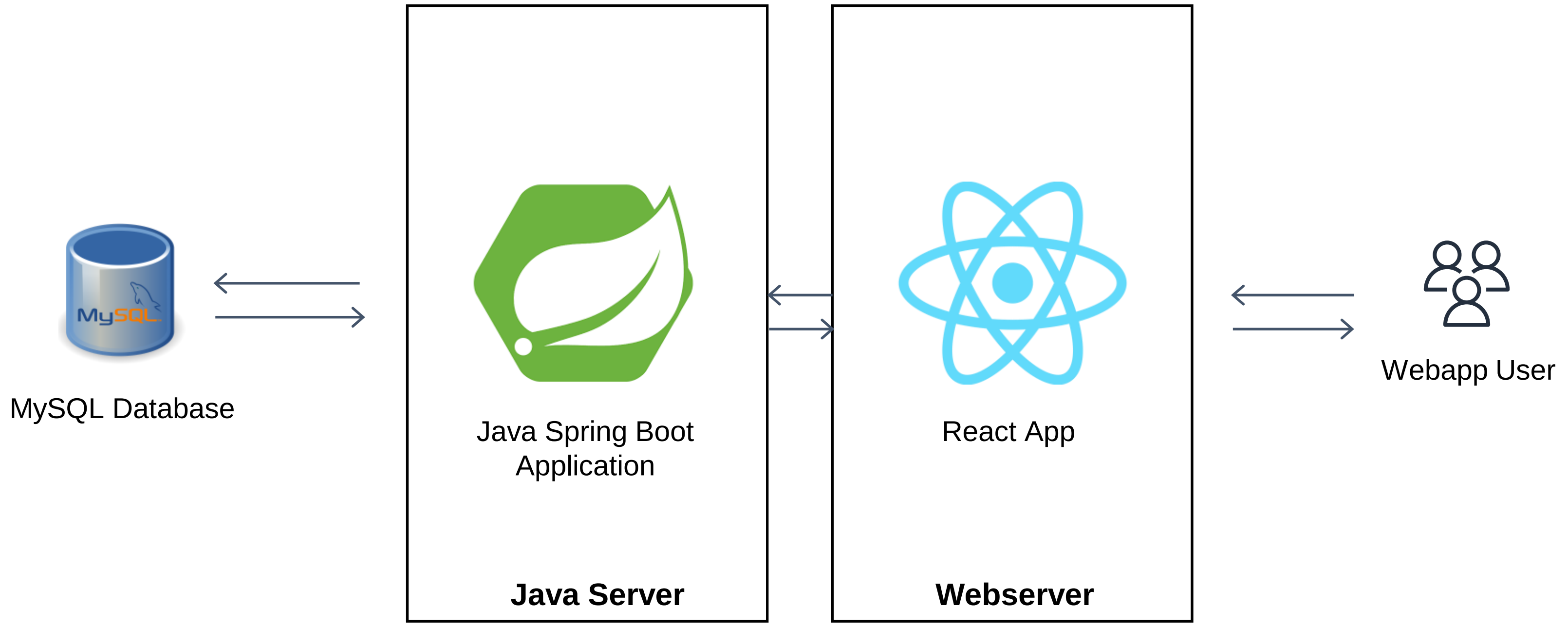
Class Exercise – Cloud Intro I [https://bit.ly/3UBmoRN]

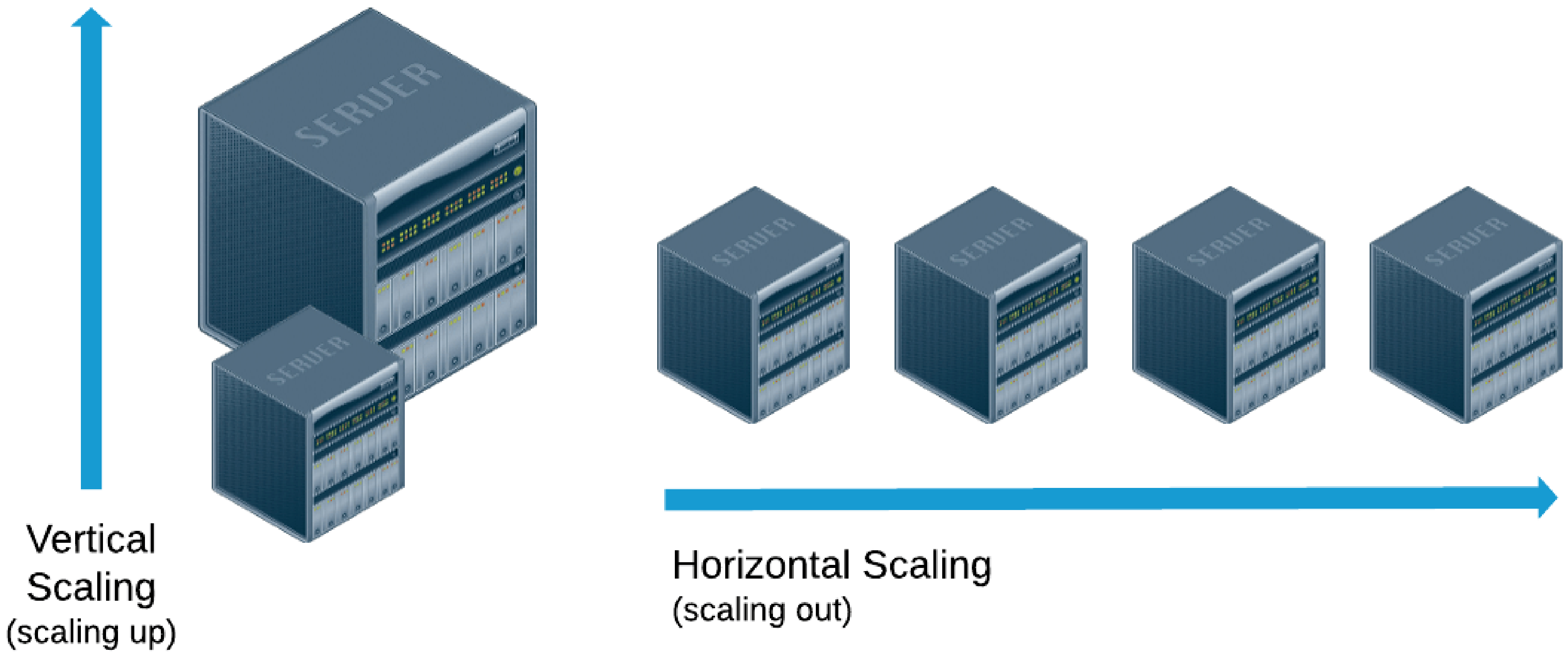


Traditional Backend-Frontend Architecture

DEMO

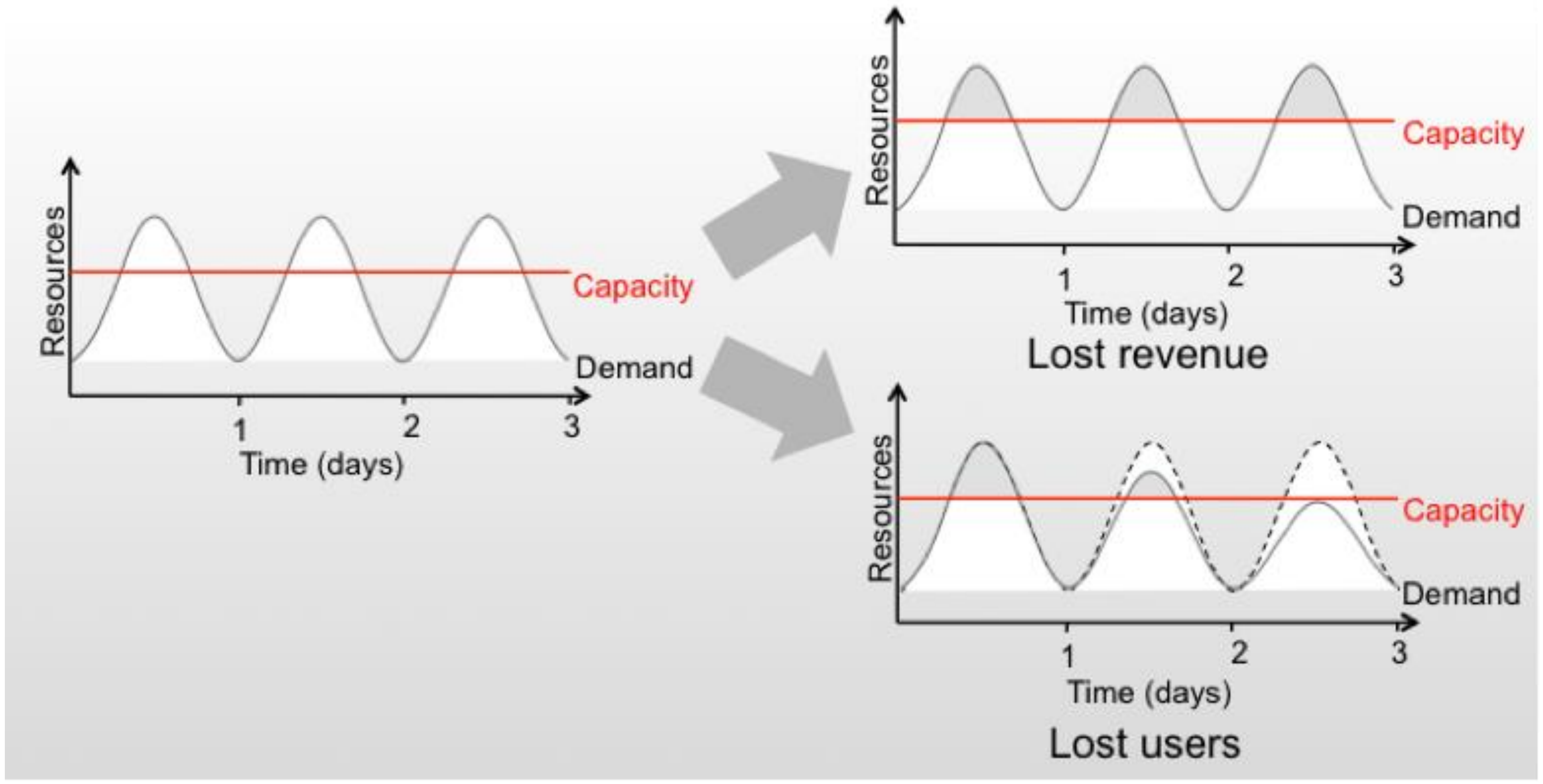
Class Exercise – Cloud Intro II [https://bit.ly/4btFnV5]



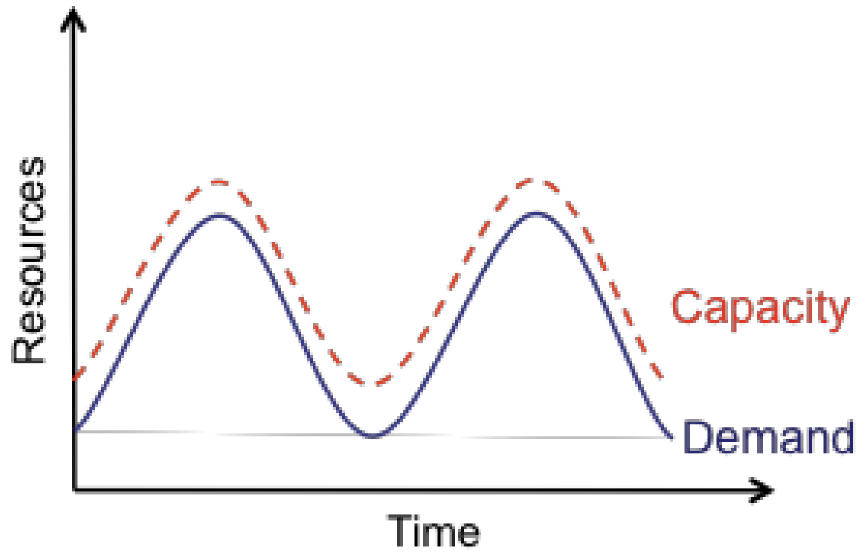


https://www.section.io/blog/scaling-horizontally-vs-vertically/

Cloud — Elasticity



Cloud — Elasticity



- elasticity in practice depends on:
  - resource granularity
  - billing granularity
  - provisioning time

Cloud Computing

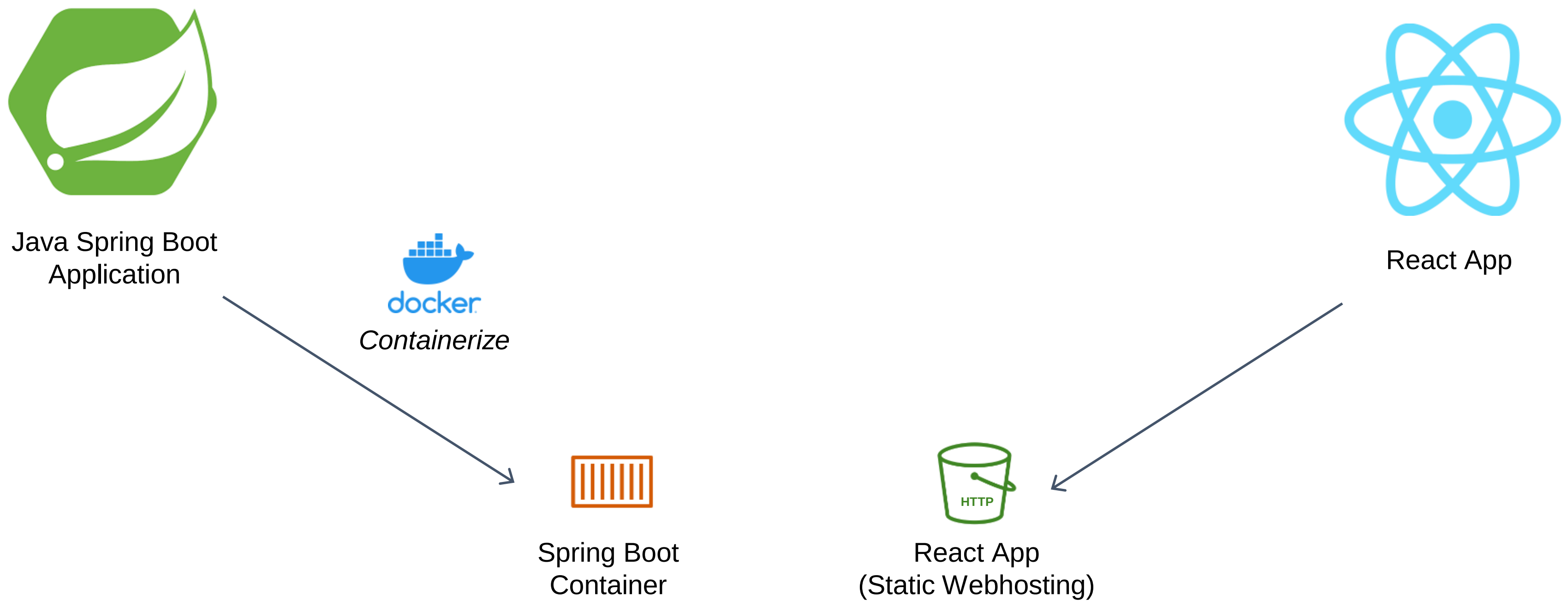
- “Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.” [NIST]

- “Illusion of infinite resources”; no up-front commitment; pay for use

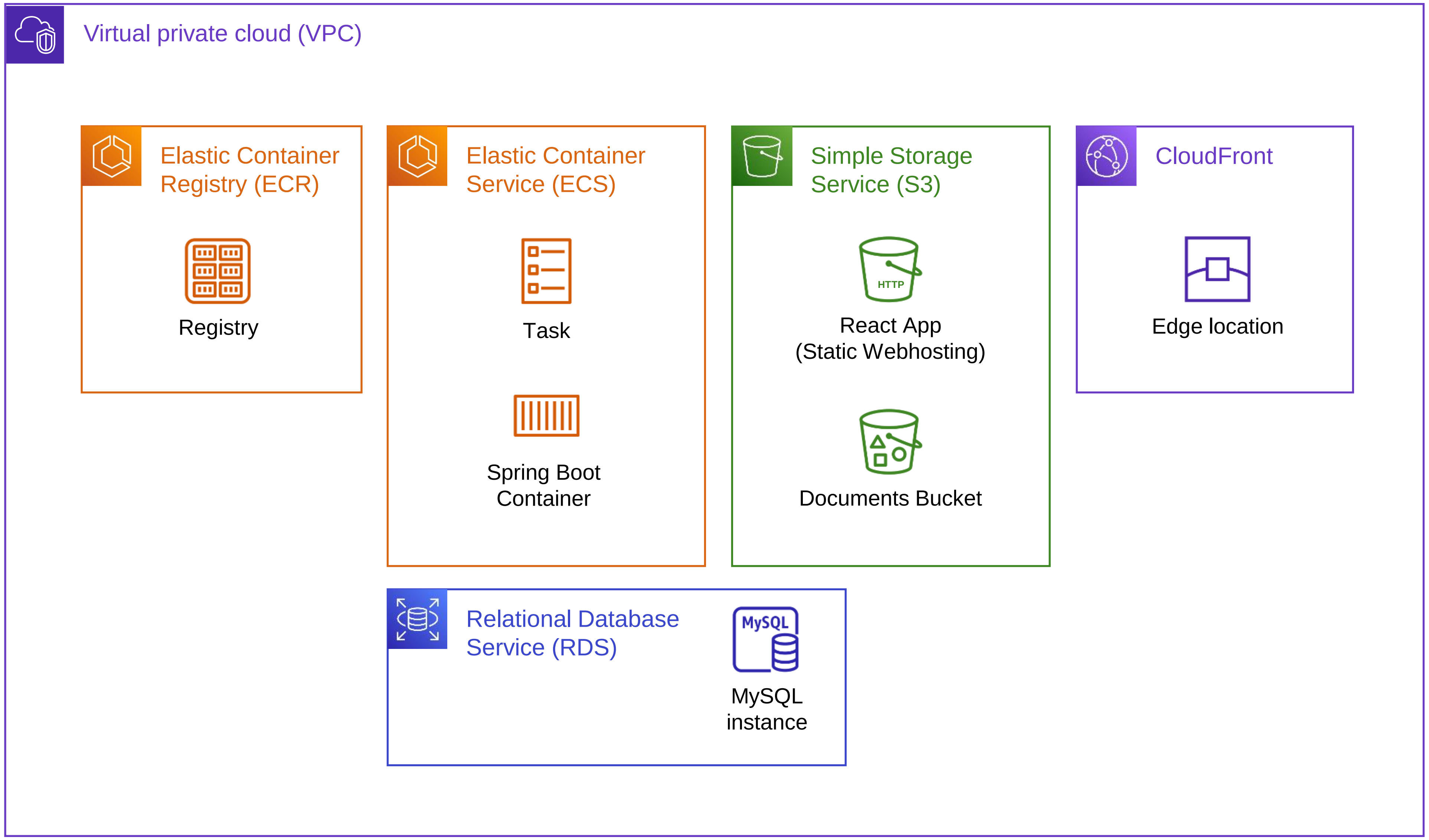
NIST Essential Characteristics

- On-demand self-service
  - Consumers can provision computing capabilities without human interaction.
- Resource pooling
  - Computing resources are pooled to serve multiple consumers.
  - Location independence.
- Rapid elasticity
  - Resources can be easily added and removed.
- Measured service [services and/or resources]
  - Metering of storage, processing, bandwidth, etc.

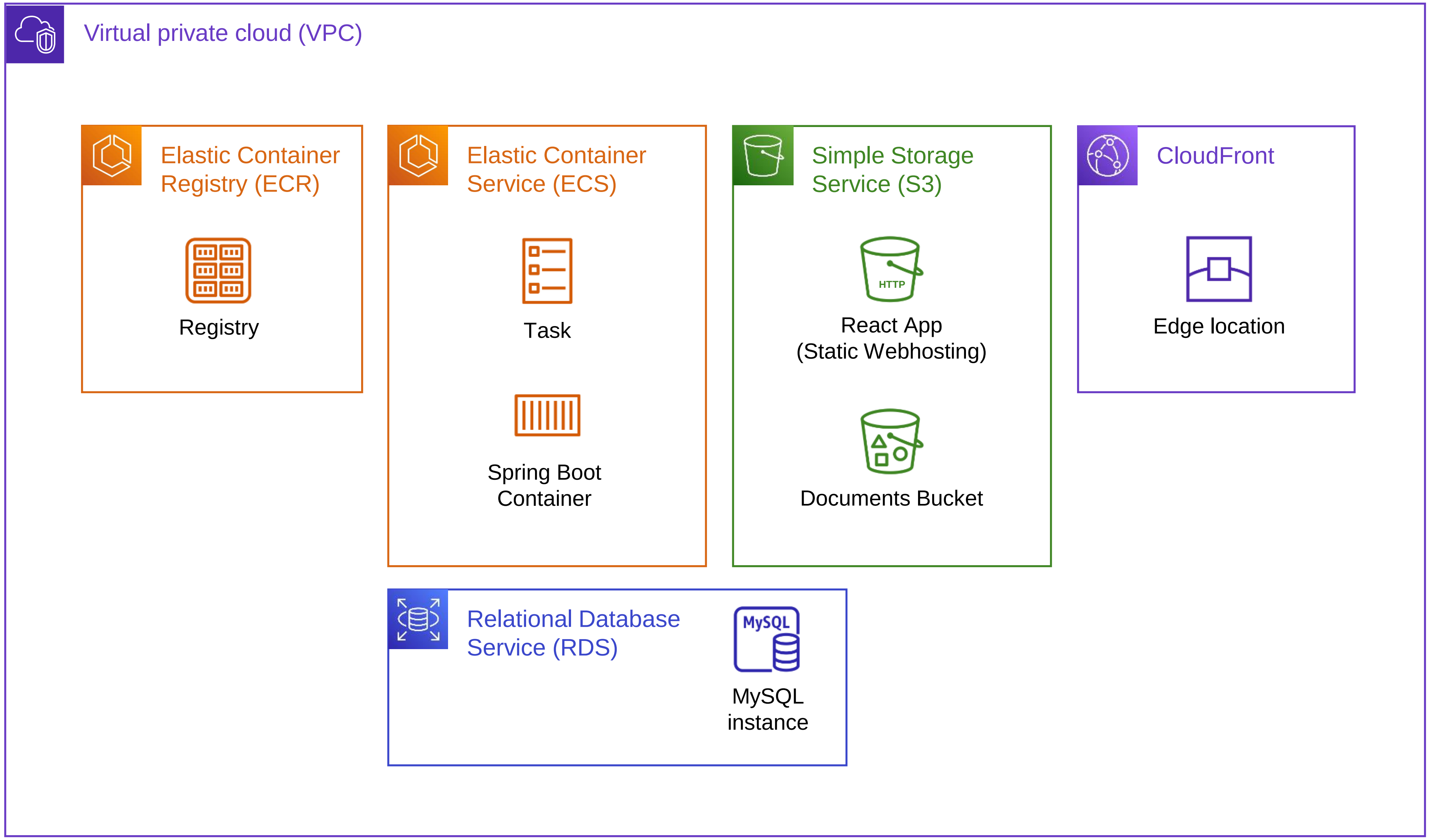
Let’s make our Spring Boot/React App scalable!



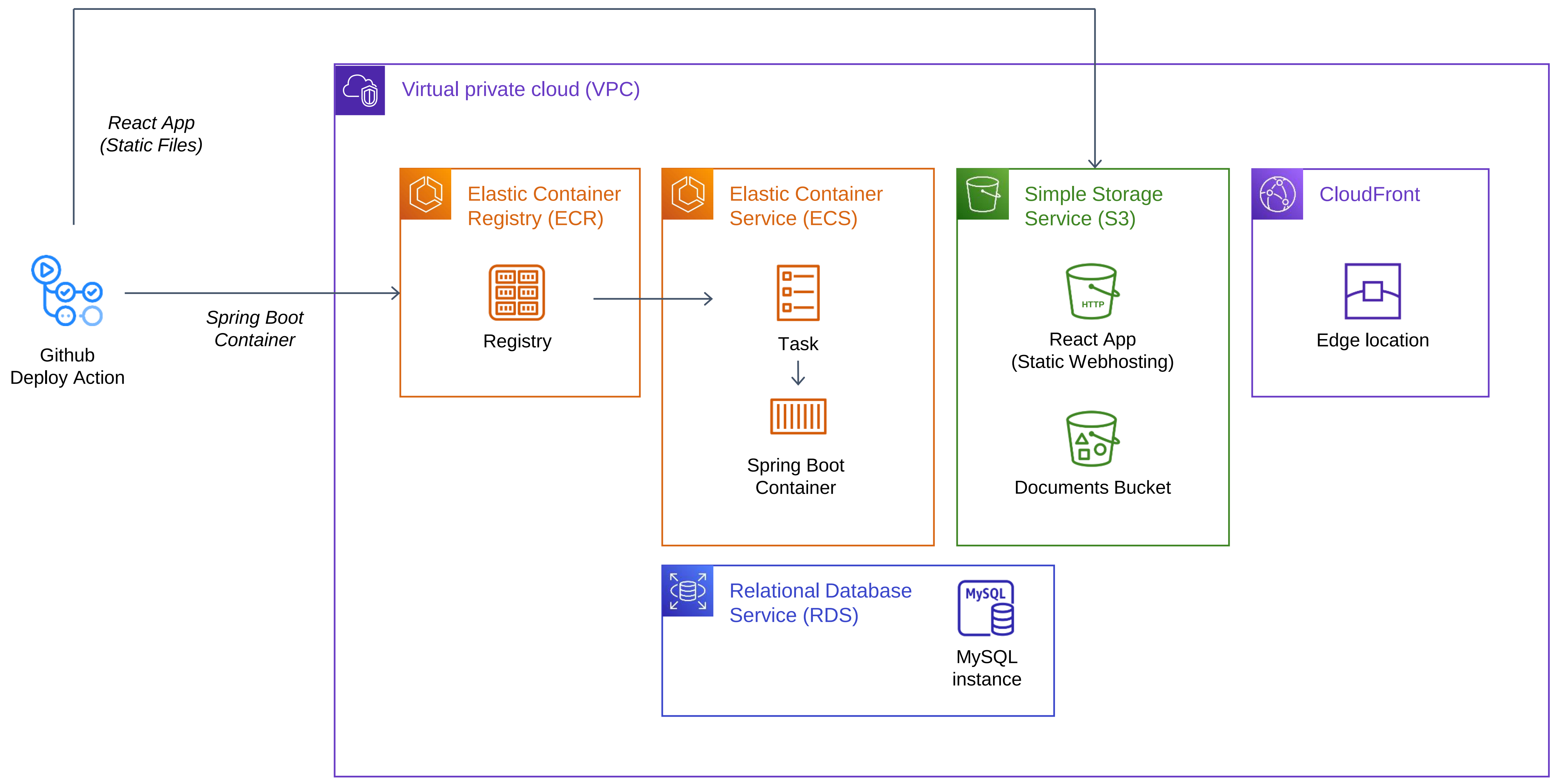
AWS Microservices Architecture


Virtual private cloud (VPC)
Elastic Container Registry (ECR)
Registry
Elastic Container Service (ECS)
Task
Spring Boot Container
Simple Storage Service (S3)
HTTP
React App (Static Webhosting)
Documents Bucket
CloudFront
Edge location
Relational Database Service (RDS)
MySQL
MySQL instance

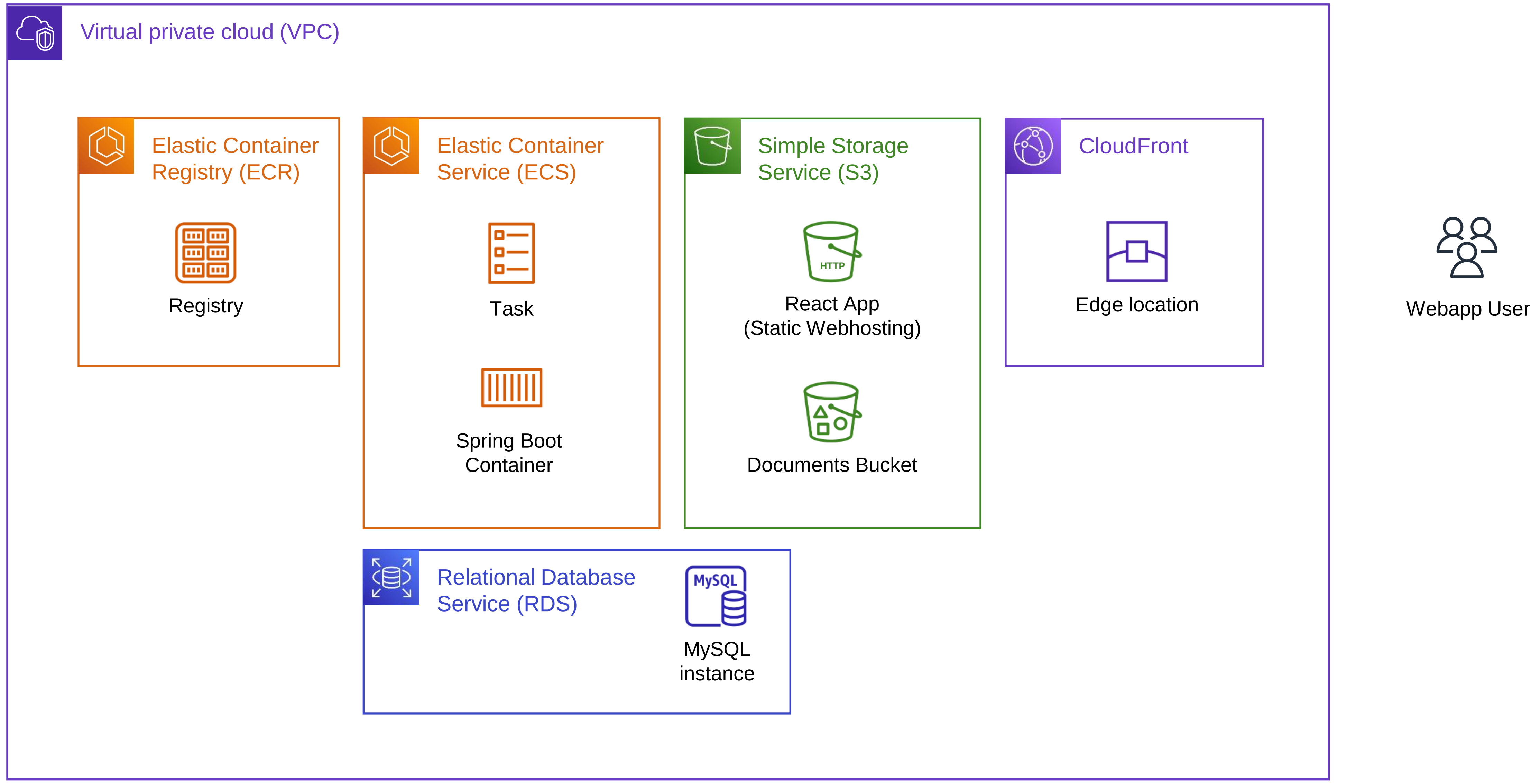
Deployment Flow


Virtual private cloud (VPC)
Elastic Container Registry (ECR)
Registry
Elastic Container Service (ECS)
Task
Spring Boot Container
Simple Storage Service (S3)
HTTP
React App (Static Webhosting)
Documents Bucket
CloudFront
Edge location
Relational Database Service (RDS)
MySQL
MySQL instance

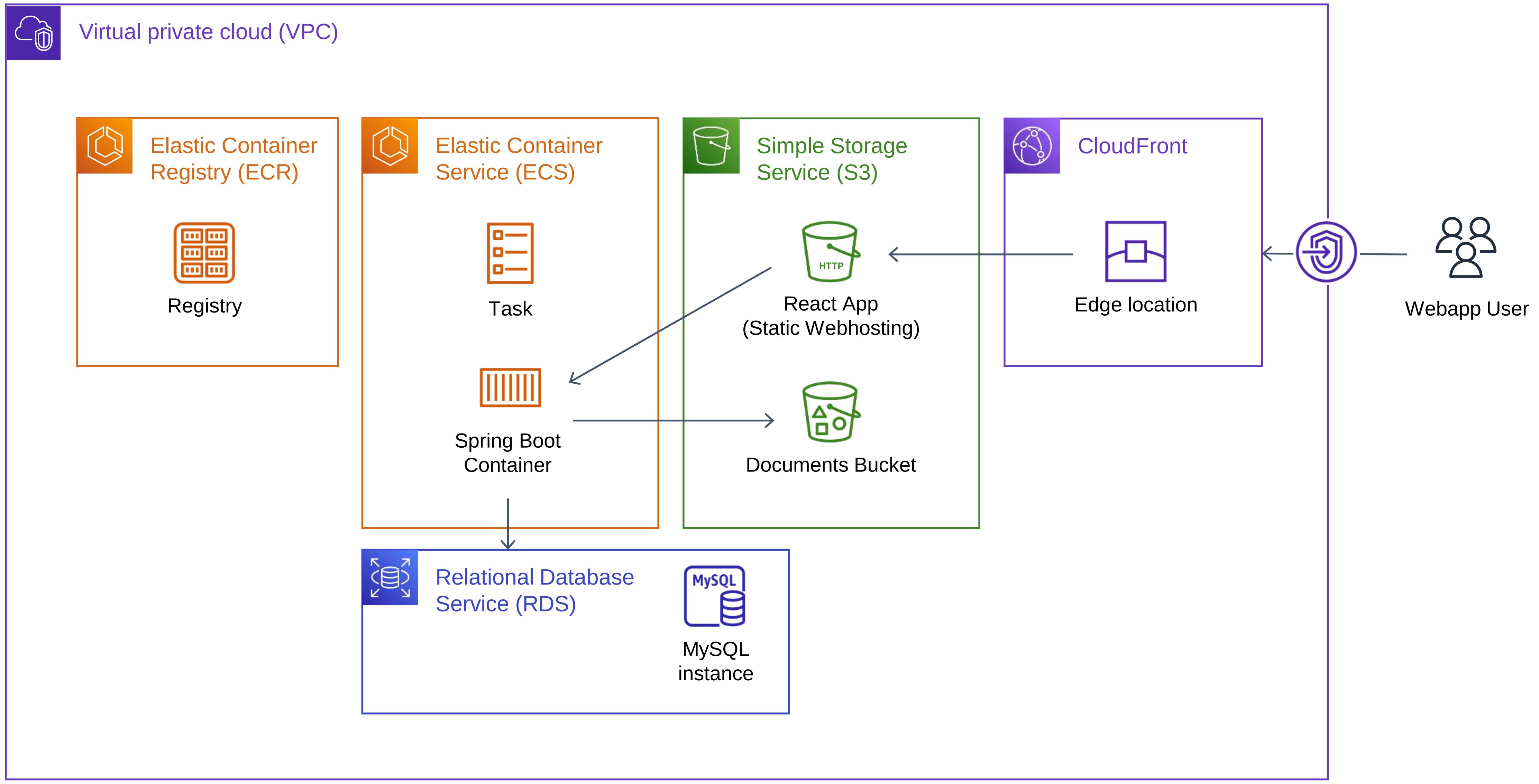
Deployment Flow



Application Flow

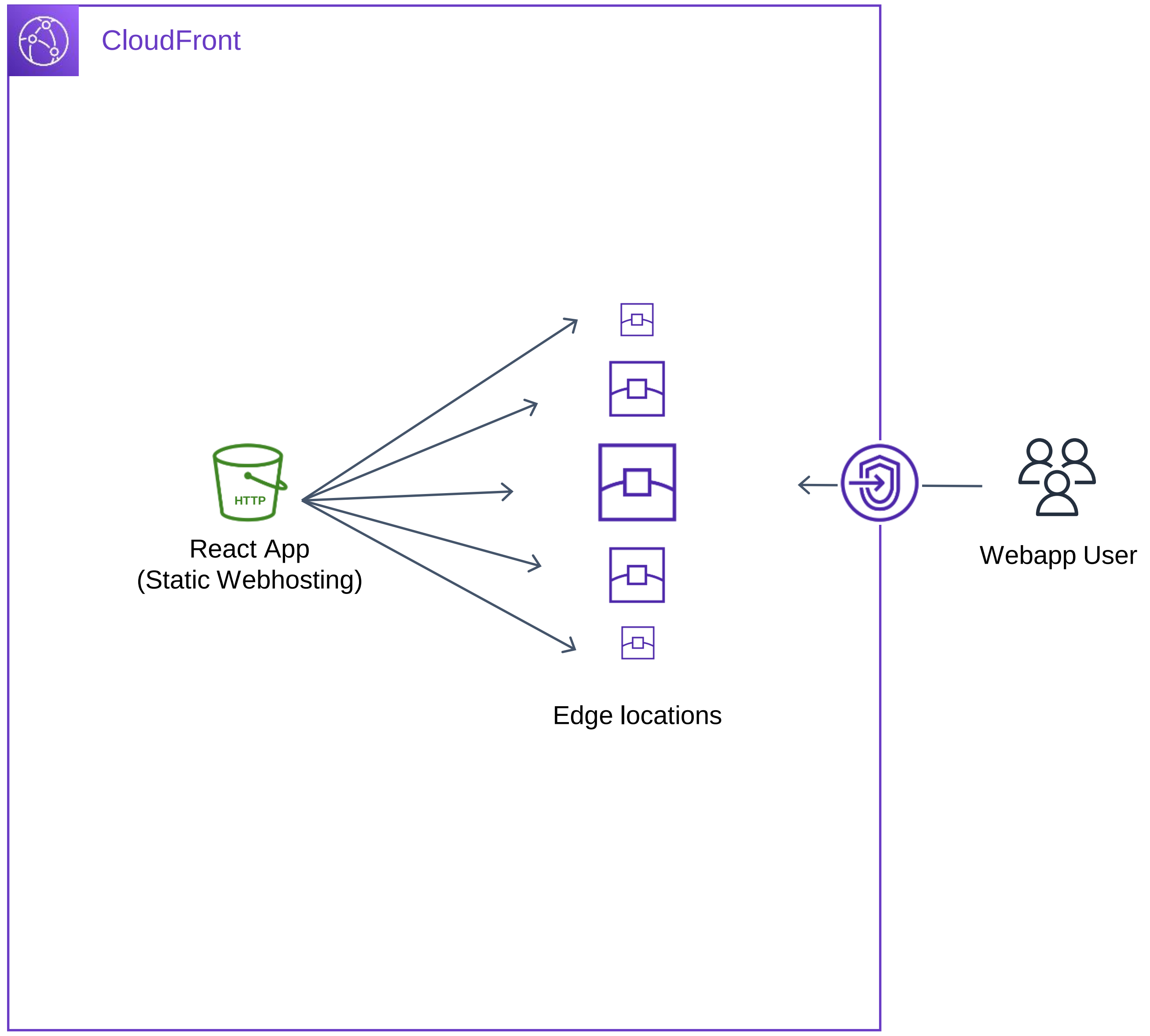

Virtual private cloud (VPC)
Elastic Container Registry (ECR)
Registry
Elastic Container Service (ECS)
Task
Spring Boot Container
Simple Storage Service (S3)
HTTP
React App (Static Webhosting)
Documents Bucket
CloudFront
Edge location
Webapp User
Relational Database Service (RDS)
MySQL
MySQL instance

Application Flow

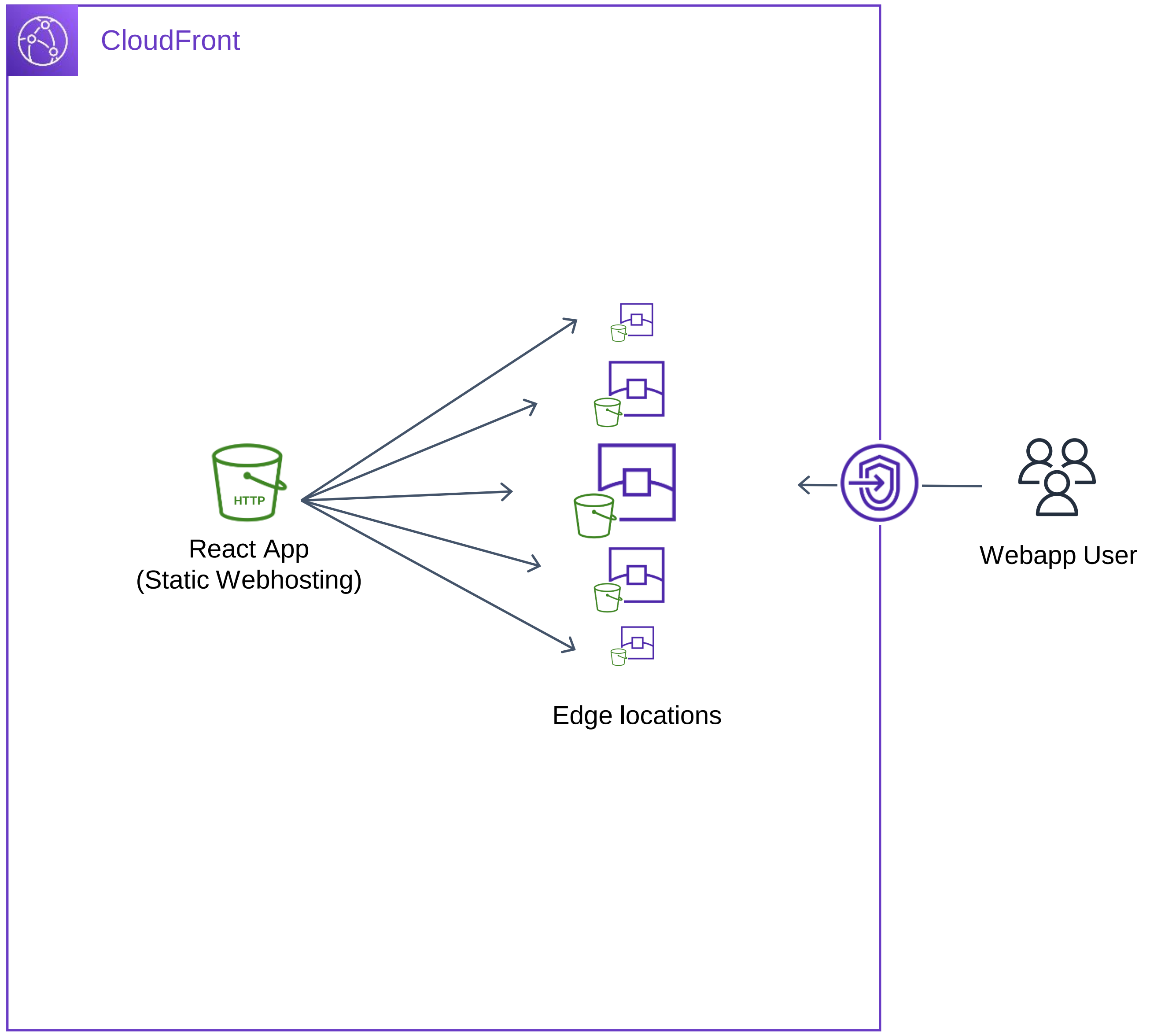


Virtual private cloud (VPC)
Elastic Container Registry (ECR)
Registry
Elastic Container Service (ECS)
Task
Spring Boot Container
Simple Storage Service (S3)
HTTP
React App (Static Webhosting)
Documents Bucket
CloudFront
Edge location
Webapp User
Relational Database Service (RDS)
MySQL
MySQL instance

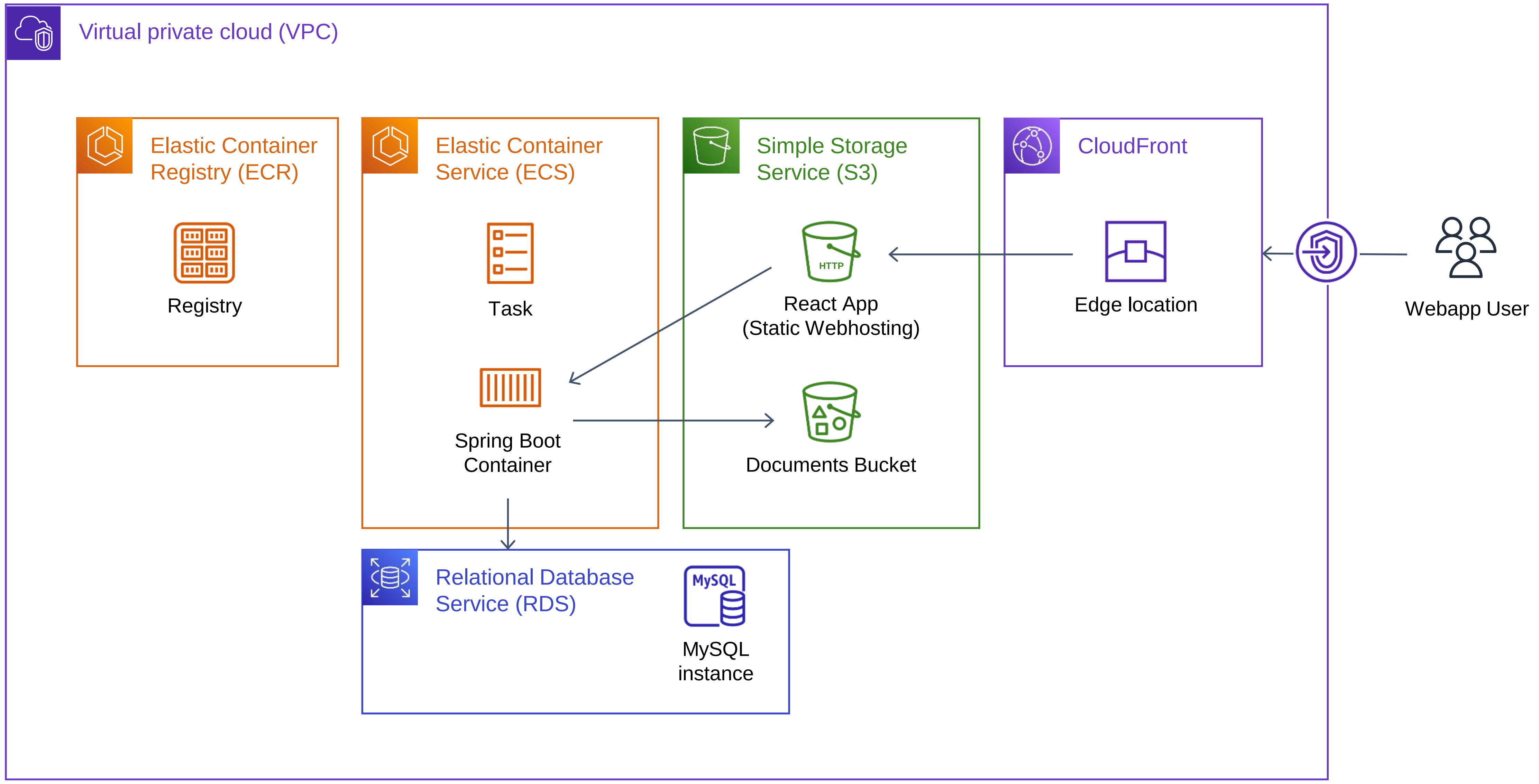
Scalability: Client App Delivery



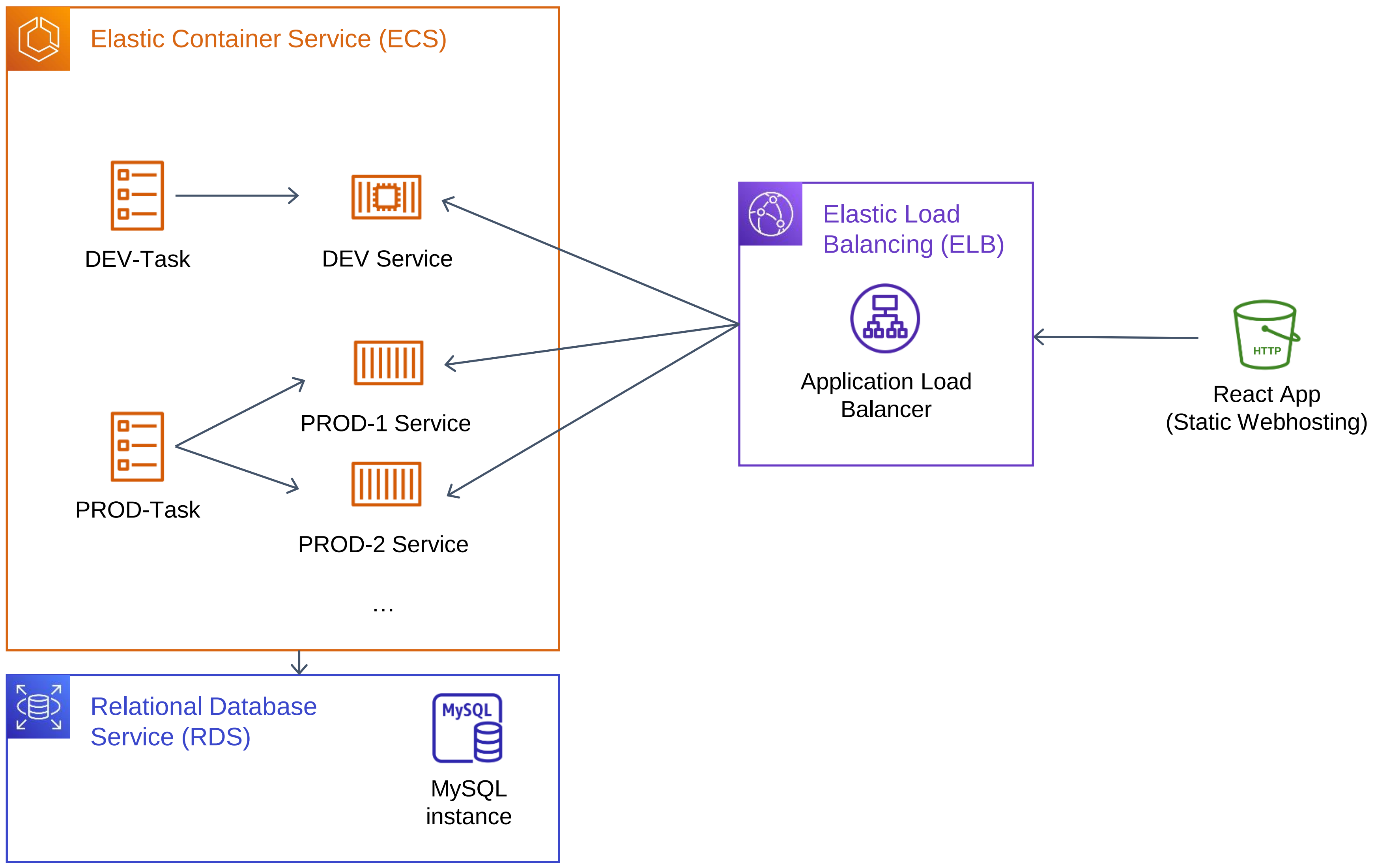
Scalability: Client App Delivery



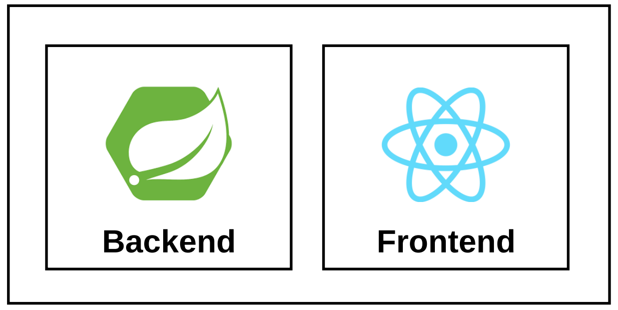
Scalability


Virtual private cloud (VPC)
Elastic Container Registry (ECR)
Registry
Elastic Container Service (ECS)
Task
Spring Boot Container
Simple Storage Service (S3)
HTTP
React App (Static Webhosting)
Documents Bucket
CloudFront
Edge location
Webapp User
Relational Database Service (RDS)
MySQL
MySQL instance

Scalability: Backend Load Balancing

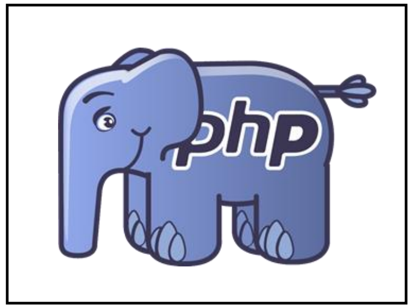

Elastic Container Service (ECS)
DEV-Task
DEV Service
PROD-1 Service
PROD-Task
PROD-2 Service
…
Elastic Load Balancing (ELB)
Application Load Balancer
HTTP
React App (Static Webhosting)
Relational Database Service (RDS)
MySQL
MySQL instance

Software Architecture Summary

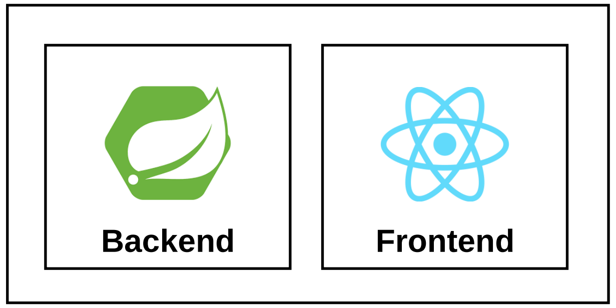


Modular Monolith

Software Architecture Summary

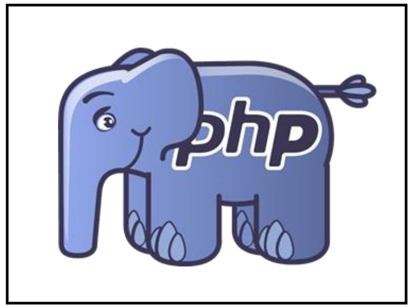


Monolith

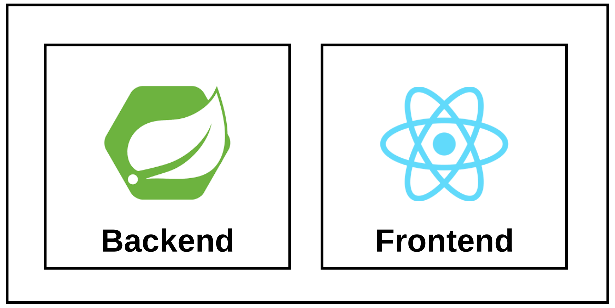


Modular Monolith

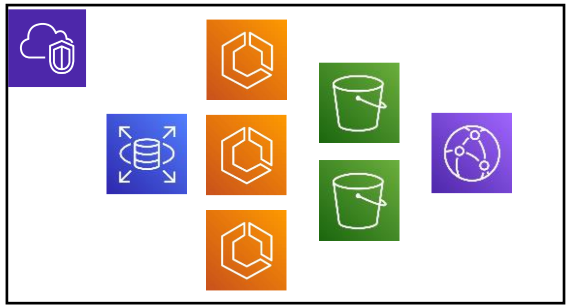
Software Architecture Summary



Monolith



Modular Monolith



Micro Services

Making apps scalable & reliable:

For each of these:
* You need to know they exist.
* And what they do.
* And how to use them.
* And how to configure them.
* And how to verify them.
* And how to monitor them.

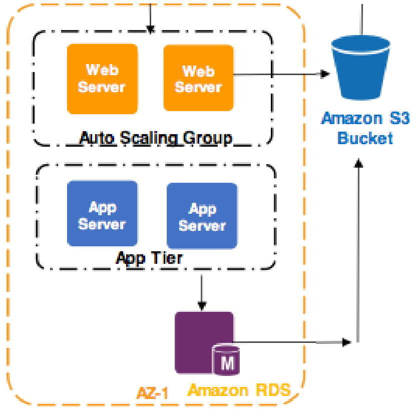
DNS

Amazon Route 53 Hosted Zone

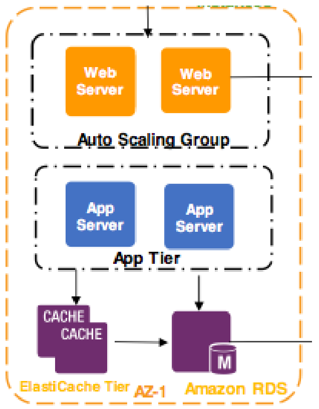
Load Balancing

Elastic Load Balancer

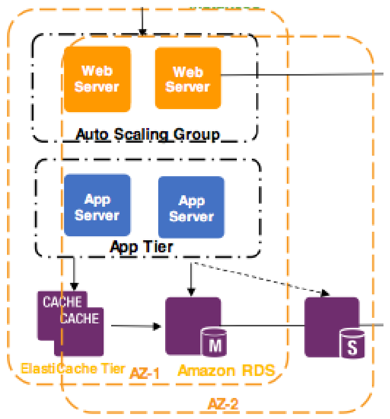
Static Caching



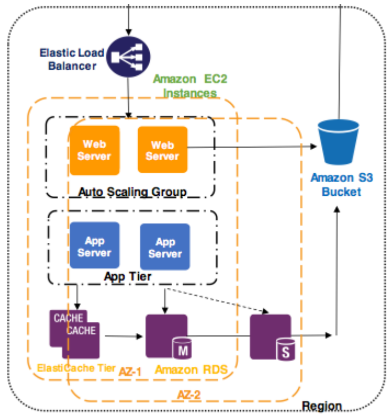
Dynamic Caching



Availability Zones



Regions



Why cloud? - Some of the benefits

- Scalability
- Reliability
- Agility
- Cost
- Security

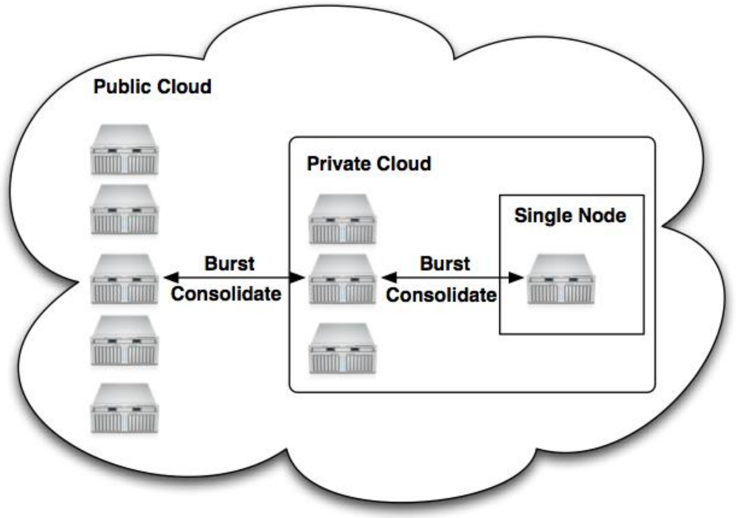
Cloud — Design and Implementation

- cloud imposes design restrictions
  - technical restriction, e.g. libraries, frameworks
- design for scalability
- design for failure (due to volatility)

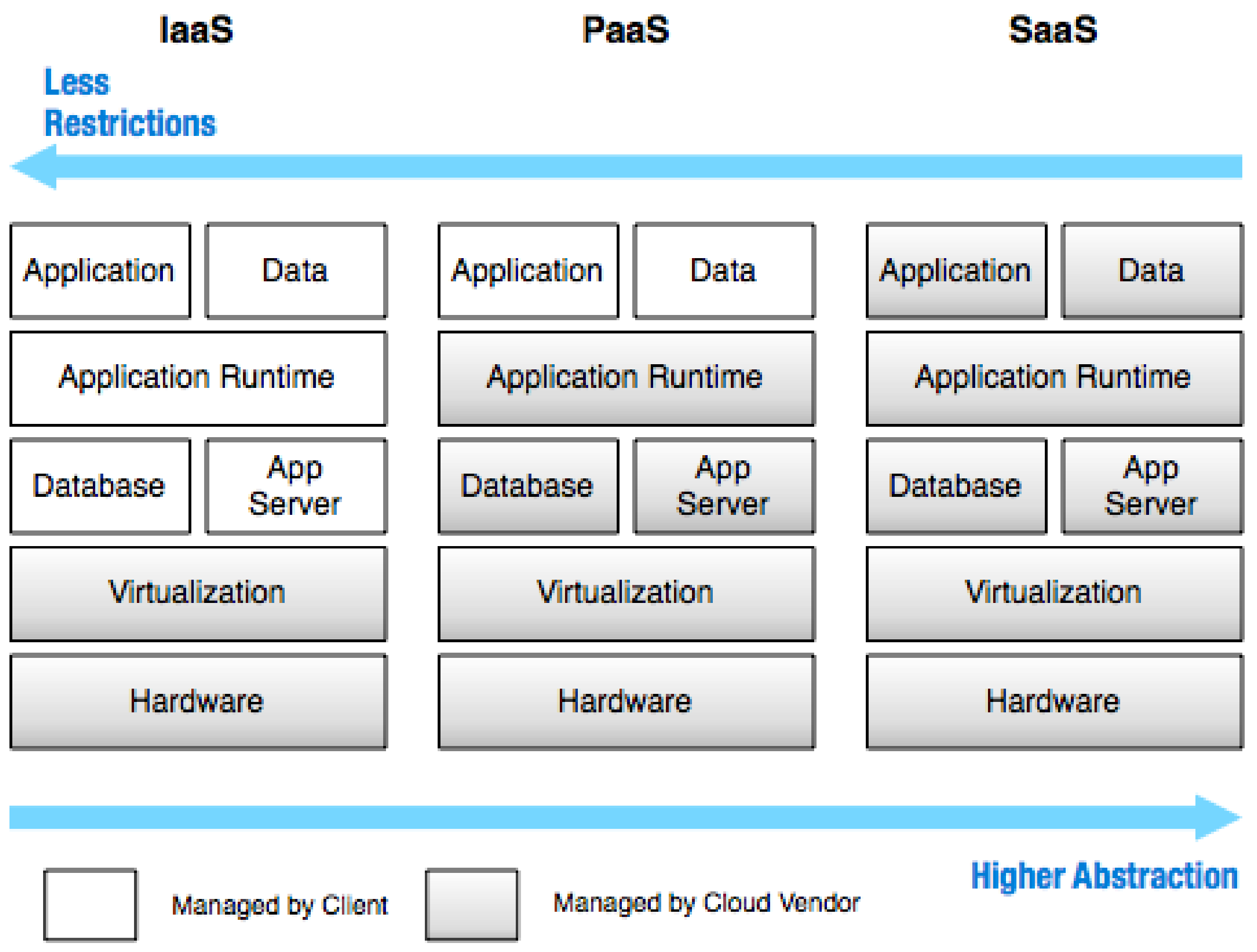
- higher need for logging and monitoring (and planning of it)

Cloud: Public vs Private vs Hybrid

- Who is hosting the cloud?
  - Public: entity providing cloud separate from consumer
  - Private: cloud principles on your own hardware
  - Hybrid: make use of multiple clouds at the same time

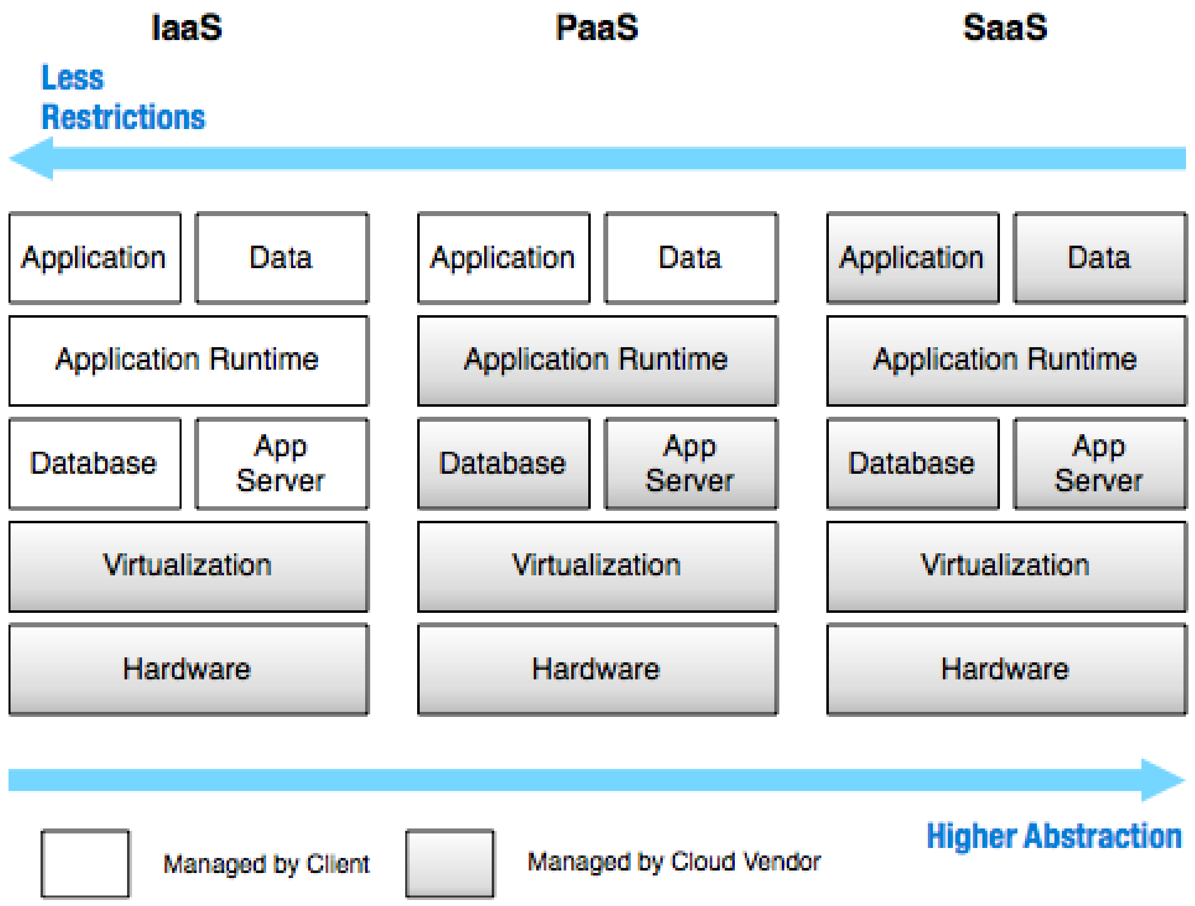


Cloud technology stack



Cloud technology stack

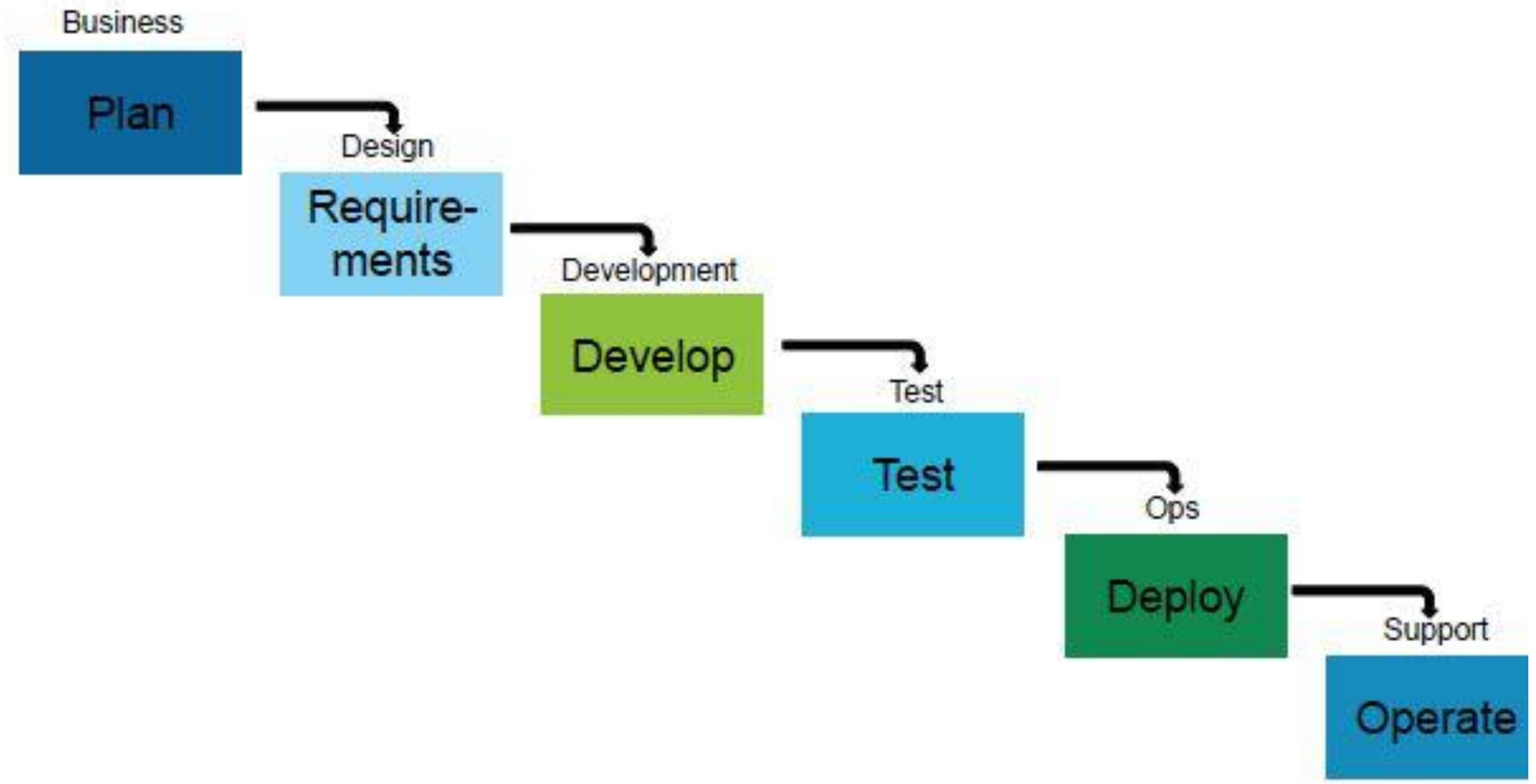
App Engine



DevOps &

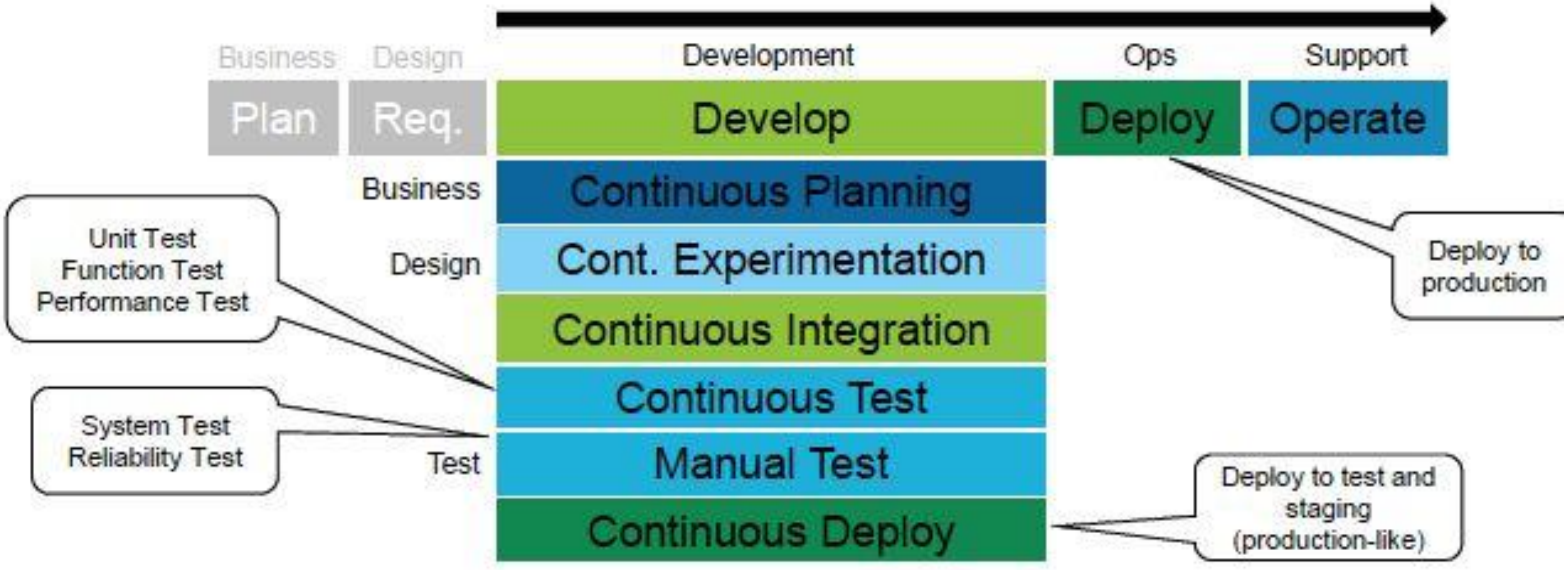
Continuous *
Integration / Delivery / Deployment

Waterfall

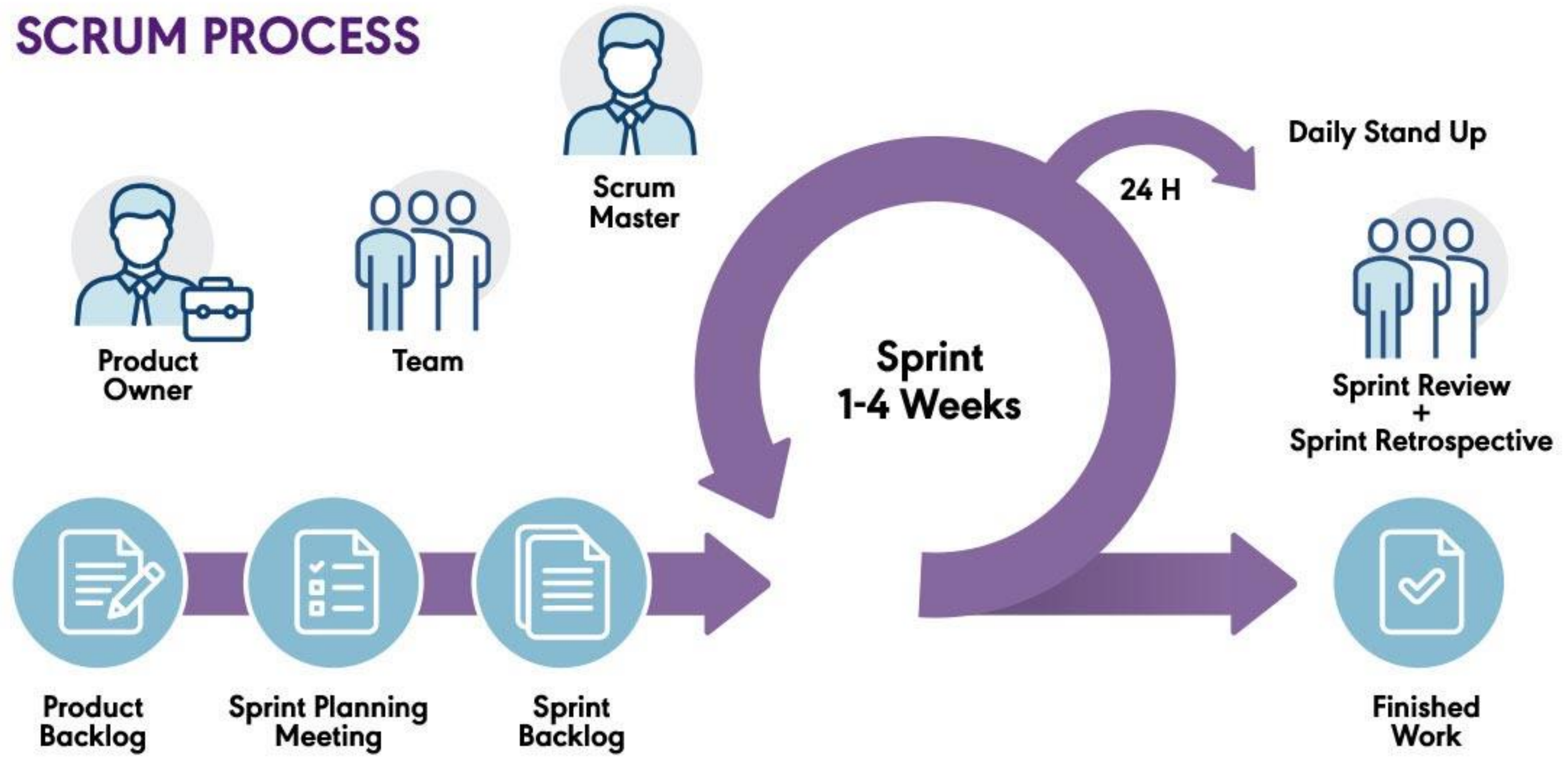


- Problems, benefits, still in use?

Continuous * (Perpetual Development)



Continuous Development



Continuous Integration (CI)

- A practice where developers **automatically** build, test, and analyze each software **change** in response to every software change committed to the source repository.

Continuous Integration (CI)

- A practice where developers **automatically** build, test, and analyze each software **change** in response to every software change committed to the source repository.

Why have CI?

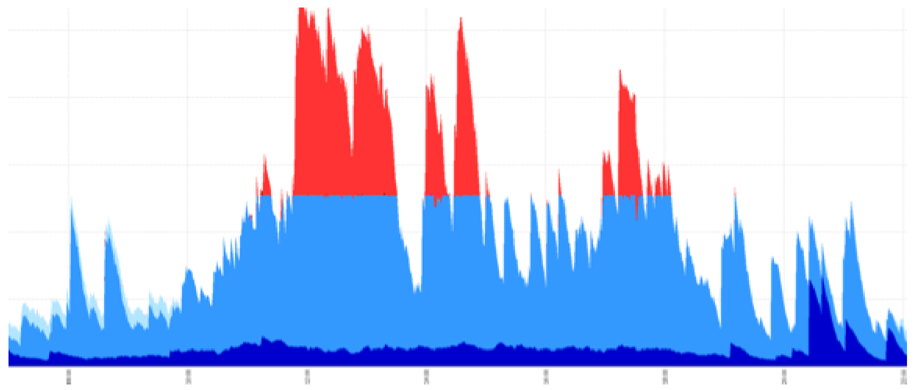
Principles

- Invest in automated tools.
- Frequent commits.
- Only commit compiling code.
- Fix broken builds immediately.
- All tests must pass.
- Run private builds.
- Pull code only from known-good configurations.

Automated Testing at Google

http://www.infoq.com/presentations/Continuous-Testing-Build-Cloud

- 10,000 devs
- 50,000 builds / day
- Monorepo
  - 20+ changes / minute
- 10 million test suites / day
  - 60 million test cases executed per day



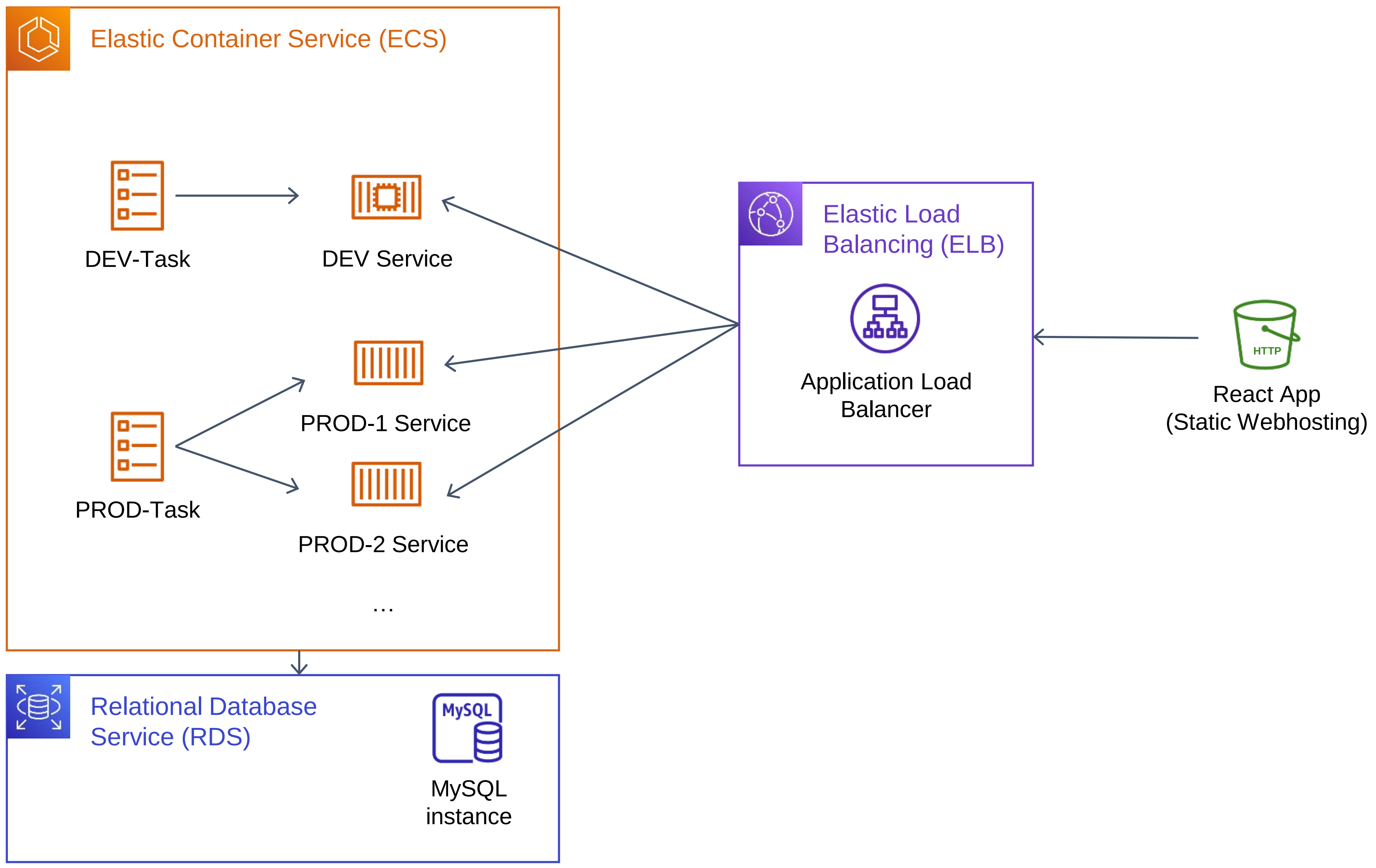
Continuous Delivery

- A practice that ensures that a software **change** can be **delivered** and ready for use by a customer in **production-like** environments.

Continuous Delivery

- Dogfooding
- Heavier infrastructure needs:
  - Test infrastructure
  - Containers helpful
    - Build & distribute virtual environment (e.g. containers like Docker)
- Why Continuous Delivery?
  - Can deploy at _any_ time (e.g., security fixes).
  - Even better feedback / more confidence.

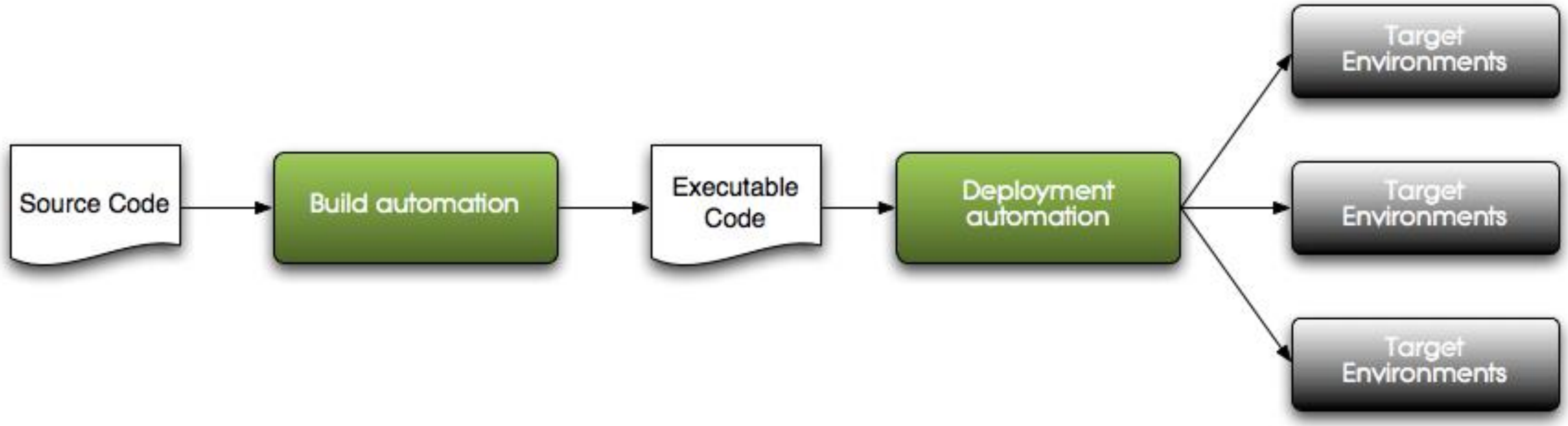
E.g. Continuous Delivery to DEV Service


Elastic Container Service (ECS)
DEV-Task
DEV Service
PROD-1 Service
PROD-Task
PROD-2 Service
...
Elastic Load Balancing (ELB)
Application Load Balancer
HTTP
React App (Static Webhosting)
Relational Database Service (RDS)
MySQL
MySQL instance

Continuous Deployment

- A practice where incremental software **changes** are automatically tested, vetted, and **deployed to production** environments.

Deployment Automation



https://dzone.com/articles/deployment-automation-vs-build

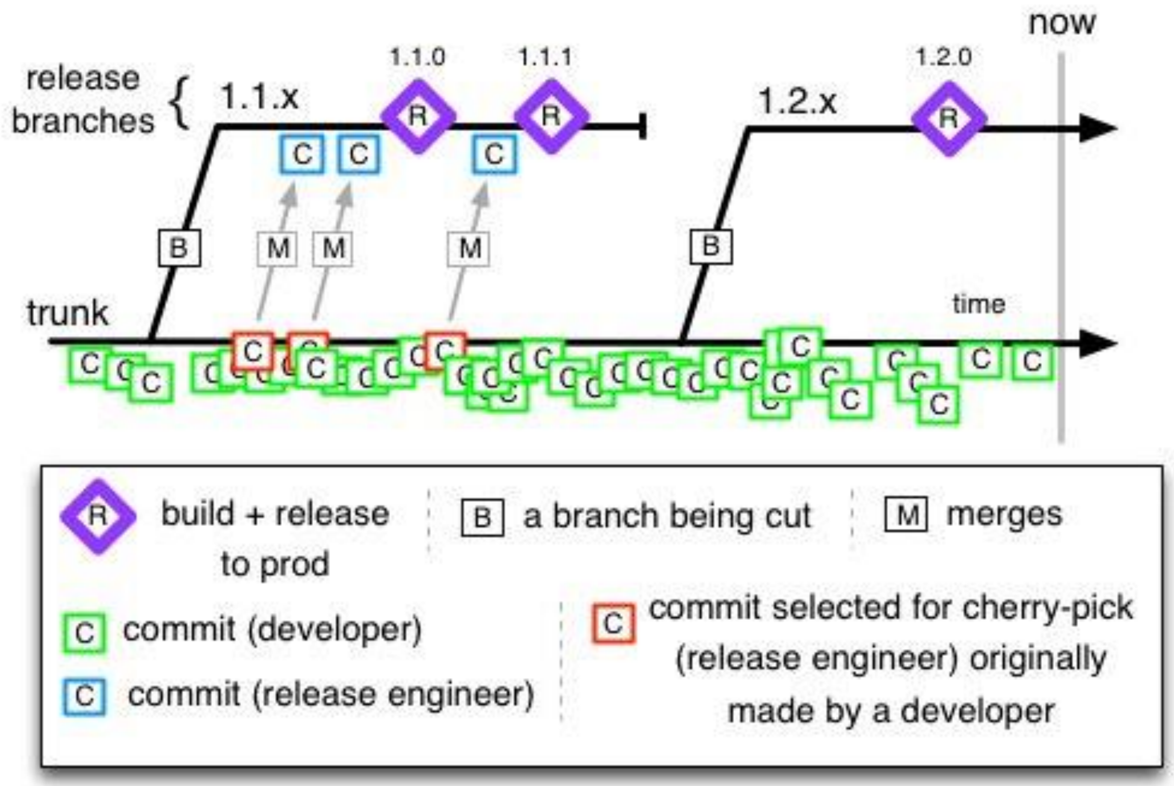
Release strategy

- Facebook is updated with hundreds of changes every day (bug fixes, new features, improvements,..)

[Optional]
Imagine you work at Facebook and want to implement 5-Star Ratings for Posts.

How would you release that feature on Facebook?

Trunk Based Development



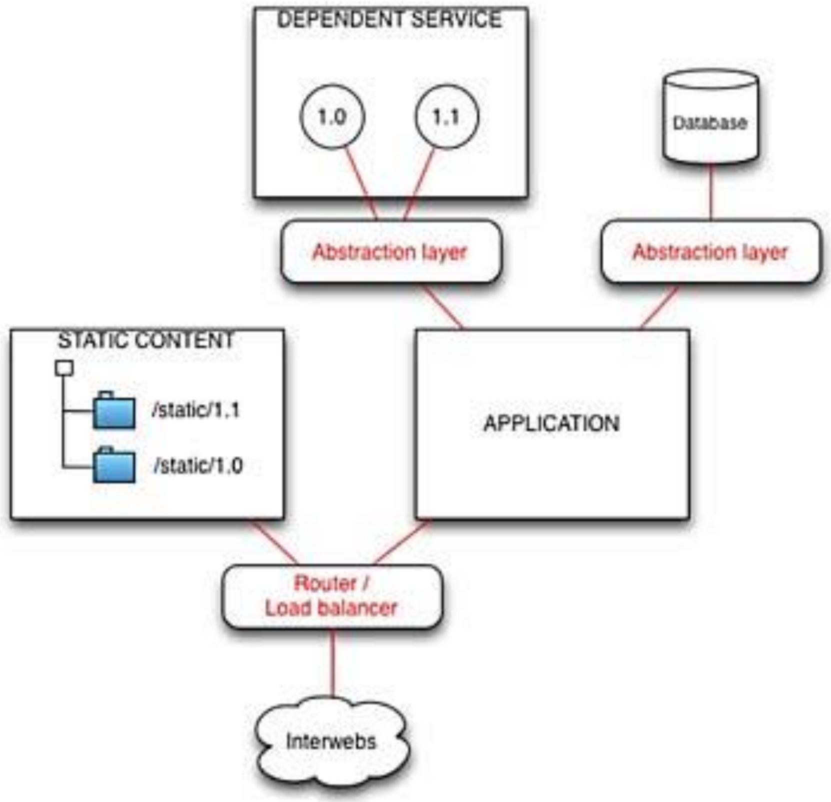
https://paulhammant.com/2013/04/05/what-is-trunk-based-development/

- e.g. facebook: new release every Tuesday; revisions cherry-picked to production as needed

Principles of low-risk releases

1. Favour incremental changes

e.g. new static content; deploy components in a side by side configuration



http://www.informit.com/articles/article.aspx?p=1833567

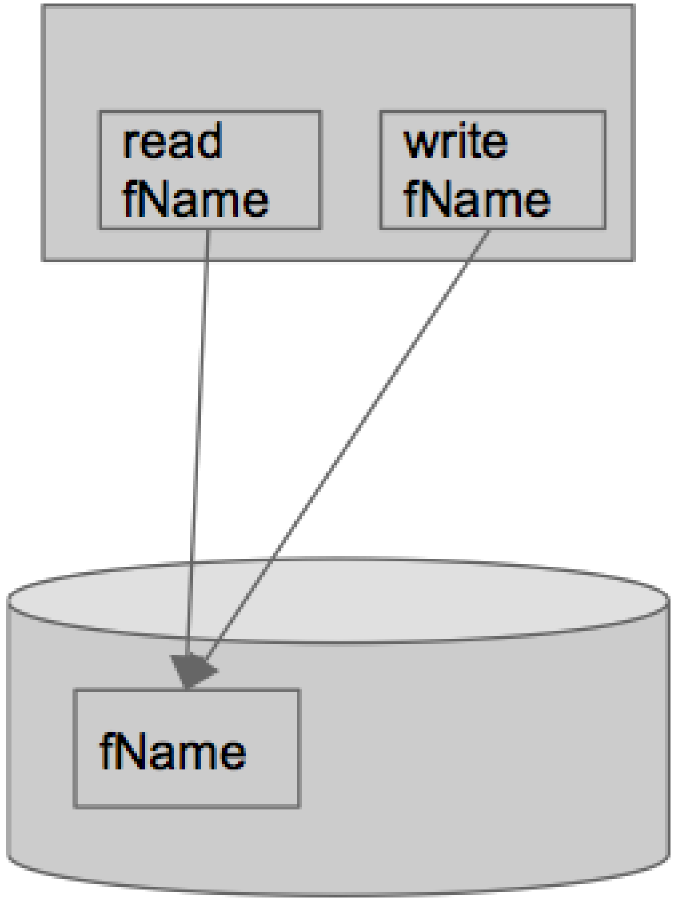
Principles of low-risk releases

1. Favour incremental changes

Expand / Contract Pattern (DB changes)

- add new objects to DB required for new release
- release new version of app, write to new objects, read from old objects, and migrate data 'lazily', roll back possible
- once new version is stable, apply contract script to finish migrating any old data and remove any old objects

Example



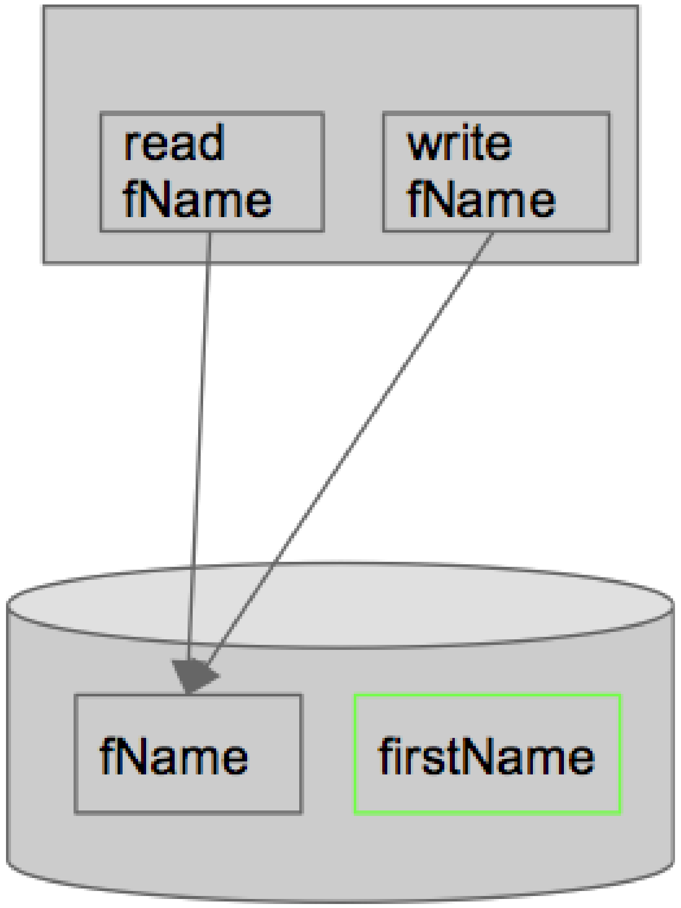
Principles of low-risk releases

1. Favour incremental changes

Expand / Contract Pattern (DB changes)

- **add new objects to DB required for new release**
- release new version of app, write to new objects, read from old objects, and migrate data 'lazily'
- once new version is stable, apply contract script to finish migrating any old data and remove any old objects

Example



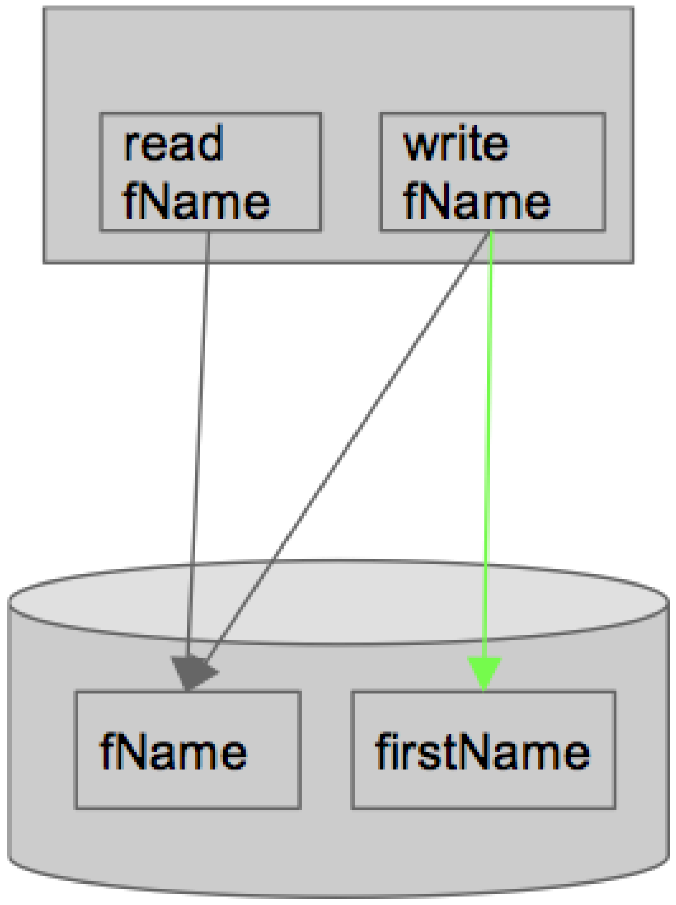
Principles of low-risk releases

1. Favour incremental changes

Expand / Contract Pattern (DB changes)

- add new objects to DB required for new release
- **release new version of app, write to new objects, read from old objects,** and migrate data 'lazily'
- once new version is stable, apply contract script to finish migrating any old data and remove any old objects

Example



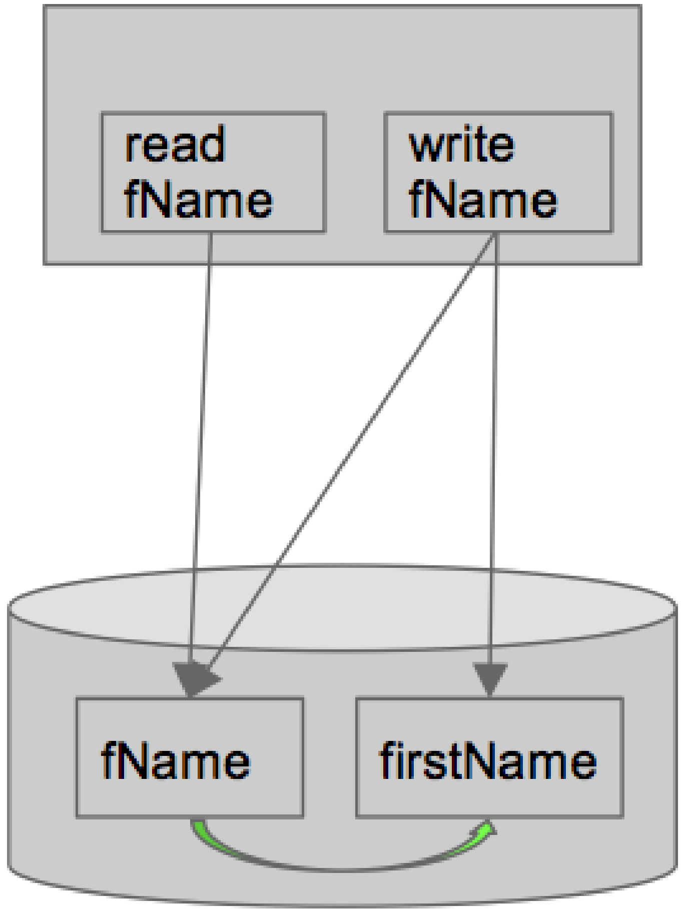
Principles of low-risk releases

1. Favour incremental changes

Expand / Contract Pattern (DB changes)

- add new objects to DB required for new release
- **release new version of app,** write to new objects, read from old objects, and **migrate data 'lazily'**
- once new version is stable, apply contract script to finish migrating any old data and remove any old objects

Example



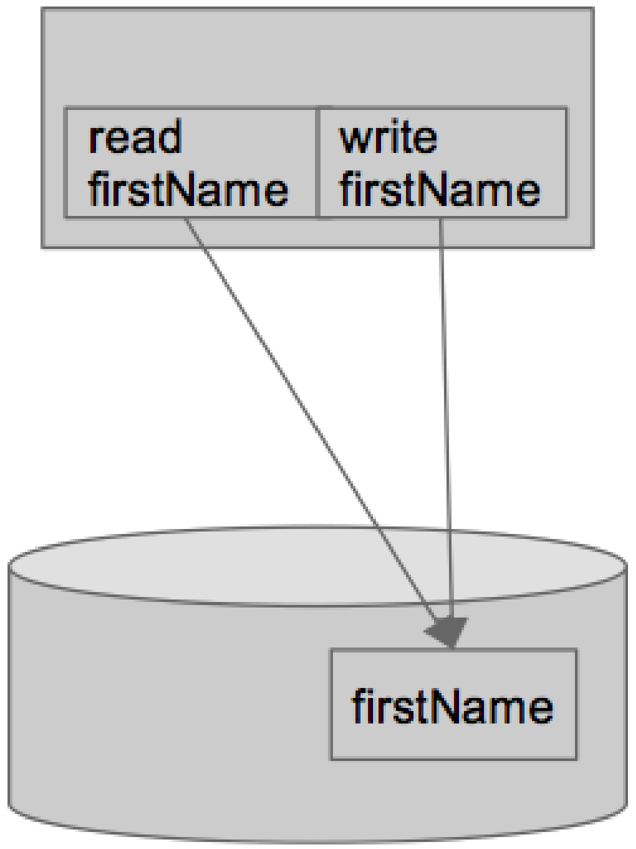
Principles of low-risk releases

1. Favour incremental changes

Expand / Contract Pattern (DB changes)

- add new objects to DB required for new release
- release new version of app, write to new objects, read from old objects, and migrate data 'lazily'
- **once new version is stable, apply contract script to finish migrating any old data and remove any old objects**

Example

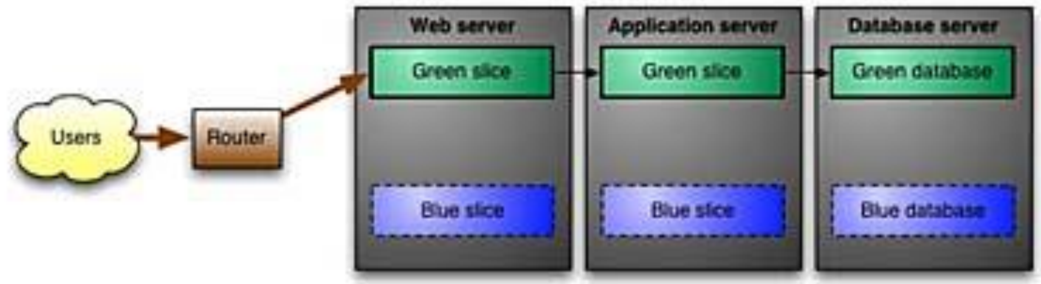


Principles of low-risk releases

1. Favour incremental changes

Green/Blue Deploy (service)

- two production environments
- big flip; router that manages user migration

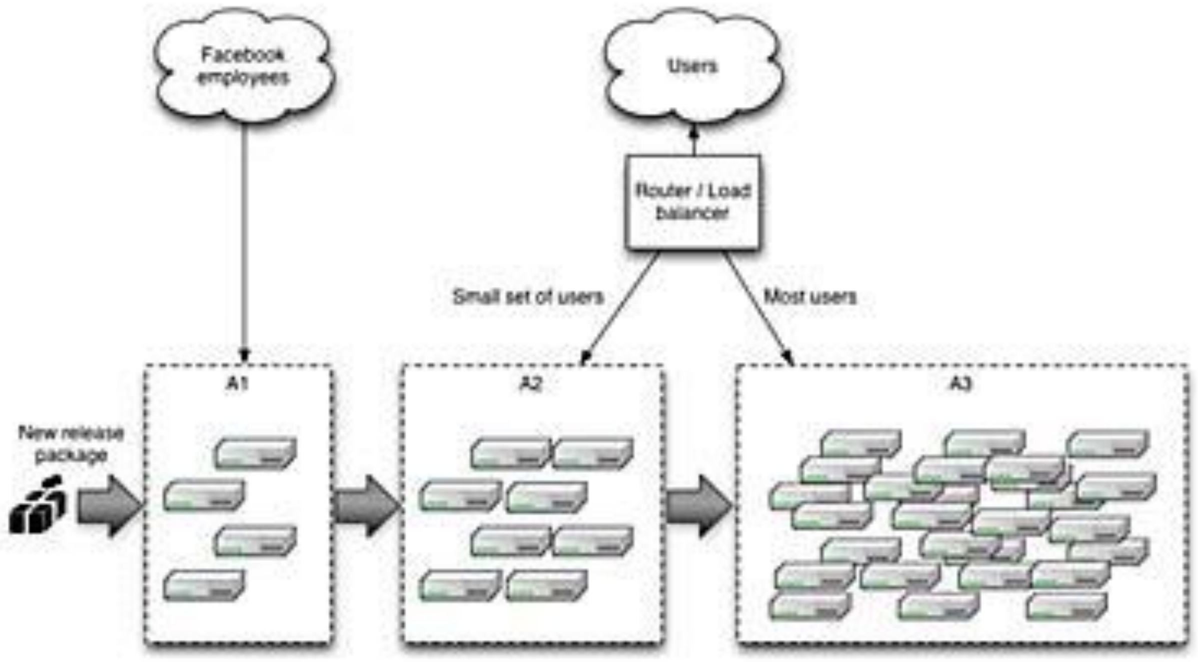


http://www.informit.com/articles/article.aspx?p=1833567

Principles of low-risk releases

1. Favour incremental changes

Canary Releasing



http://www.informit.com/articles/article.aspx?p=1833567

Principles of low-risk releases

2. Decouple deploying from releasing

- *Deployment* is what happens when you install some version of your software into a particular environment (the production environment is often implied). *Release* is when you make a system or some part of it (for example, a feature) available to users.

Principles of low-risk releases

2. Decouple deploying from releasing

Dark Launch

- Release changes without any user-facing elements exposed.
- Can help eliminate the need to support long-running release branch. Reduces merge issues. Improve disaster recovery
- Can simultaneously direct traffic through previous and dark-launched code for load testing.

Dark launches (facebook):

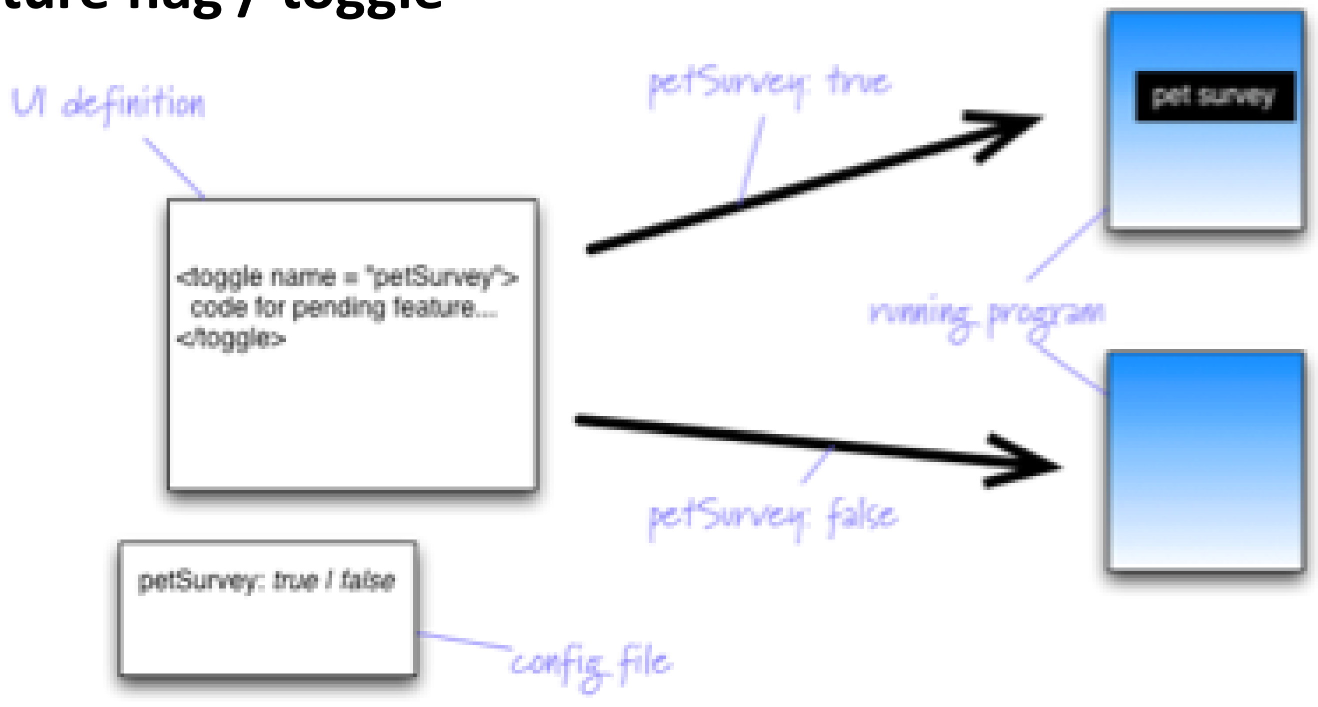
- *The secret for going from zero to seventy million users overnight is to avoid doing it all in one fell swoop. We chose to simulate the impact of many real users hitting many machines by means of a "dark launch" period in which Facebook pages would make connections to the chat servers, query for presence information and simulate message sends without a single UI element drawn on the page. With the "dark launch" bugs fixed, we hope that you enjoy Facebook Chat now that the UI lights have been turned on.*

- *https://github.com/CSC-DevOps/Course/blob/master/Readings/Deployment.md*

Principles of low-risk releases

2. Decouple deploying from releasing

Feature flag / toggle



- Gatekeeper can direct specific subsets of traffic to newly-launched code to gather data/feedback.

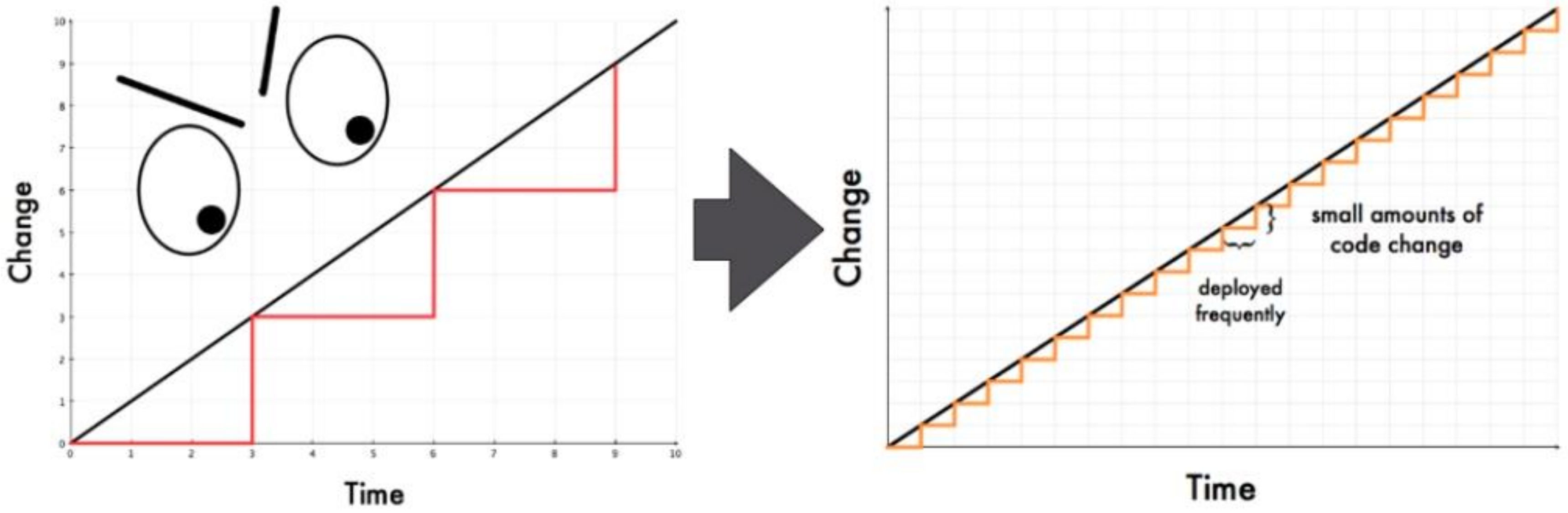
http://www.informit.com/articles/article.aspx?p=1833567

http://www.informit.com/articles/article.aspx?p=1833567

Principles of low-risk releases

3. Focus on reducing batch size

- Deploy new features to users quickly.
- Enable responsive defect resolution.
- Smaller delta == smaller faults.
- Releasing loses ‘dark art’ status.



Principles of low-risk releases

4. Optimize for resilience

- *…the ability to restore your system to a baseline state in a predictable time is vital not just when a deployment goes wrong, but also as part of your disaster-recovery strategy…*

Have a Plan B

- No, *really*
- Culture is fundamental to identifying, fixing, and recovering from large distributed faults
  - Do you know if there’s a problem?
  - Can you figure out what it is?
  - Can you find out who should fix it?
  - Fix it (we’re good at this)
  - Do we know how to deploy it?
  - Can we validate the problem is fixed?

Testing Plan B

- **CHAOS MONKEYS**
- Fail often, tolerate failure
- Learn with scale, not models
- Netflix Simian Army:
  - Latency
  - Doctor
  - Janitor
  - Security
  - …

NETFLIX

http://techblog.netflix.com/2011/07/netflix-simian-army.html

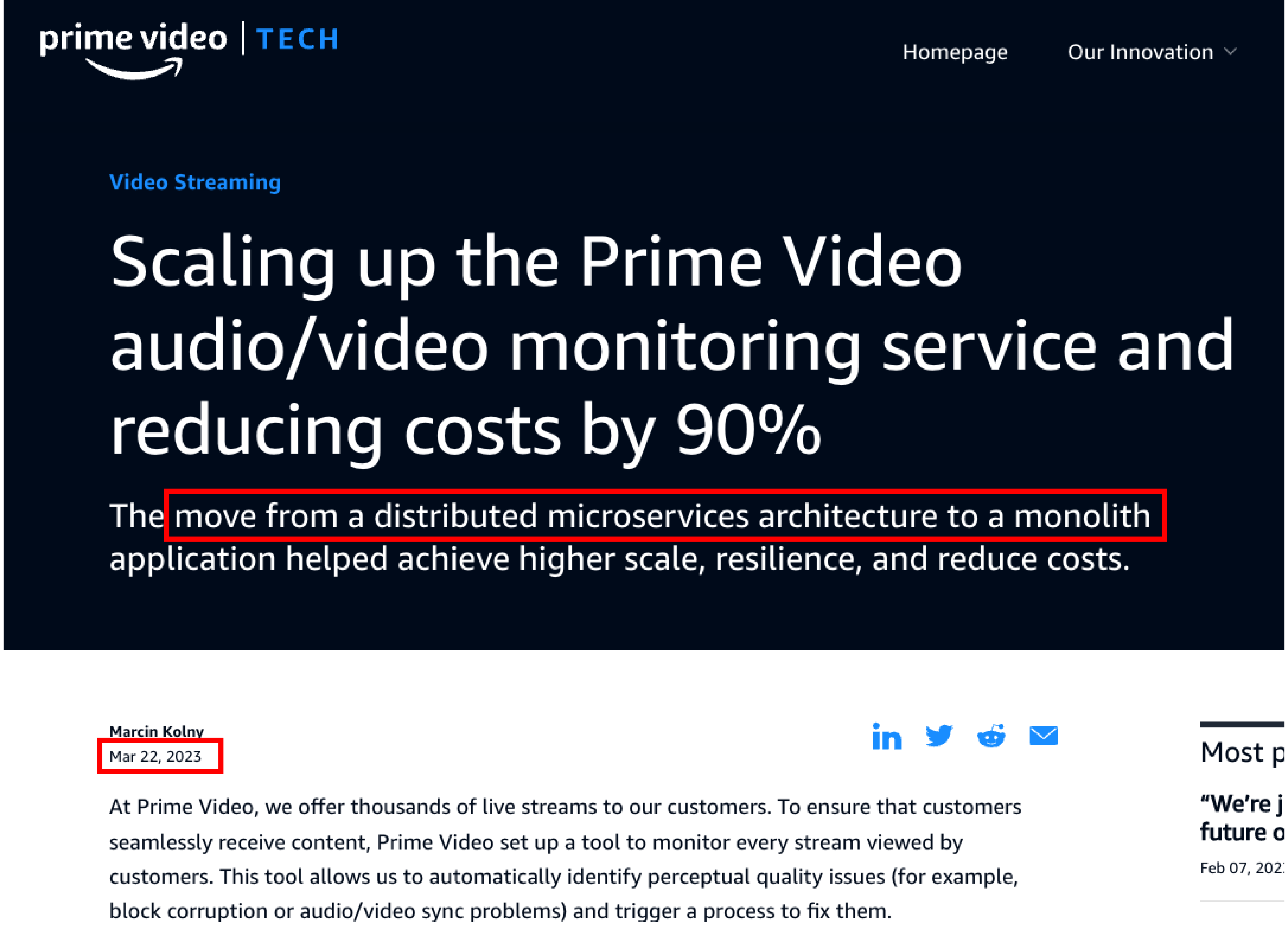
http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html

Chaos Monkey

"Unleashing a wild monkey with a weapon in your data centre (or cloud region) to randomly shoot down instances and chew through cables"

http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html

Final note



Video Streaming

Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%

The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.

Marcin Kolny
Mar 22, 2023

At Prime Video, we offer thousands of live streams to our customers. To ensure that customers seamlessly receive content, Prime Video set up a tool to monitor every stream viewed by customers. This tool allows us to automatically identify perceptual quality issues (for example, block corruption or audio/video sync problems) and trigger a process to fix them.

Most p

"We're j
future o

Feb 07, 202

https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90

Final note

- There is no generalizable, best software architecture or deployment infrastructure. It is about **choosing the best trade-off** for your use case.

Quiz

Assess whether the following statement is true or false:
"Deployments are always successful when using an automated system."

A) True
B) False

Quiz

SaaS is most beneficial for which of the following scenarios?

A) When the organization requires flexible access to computing resources
B) When the organization needs to develop and maintain applications
C) When the organization wants ready-to-use application software

Quiz

Assess whether the following statement is true or false:
“Continuous Deployment requires a substantial upfront investment in automation and testing.”

A) True
B) False

Quiz

What can you do to lower the risk of failures when releasing a new version of an application? For each of the following statements assess whether it would help to lower the risk (true) or not (false).

A) Ensure that when you deploy a new version of your application, you also make it available to users right away
B) Deploy the new version of your application side by side with the old version before switching over.
C) Reduce the batch size and deploy more frequently.