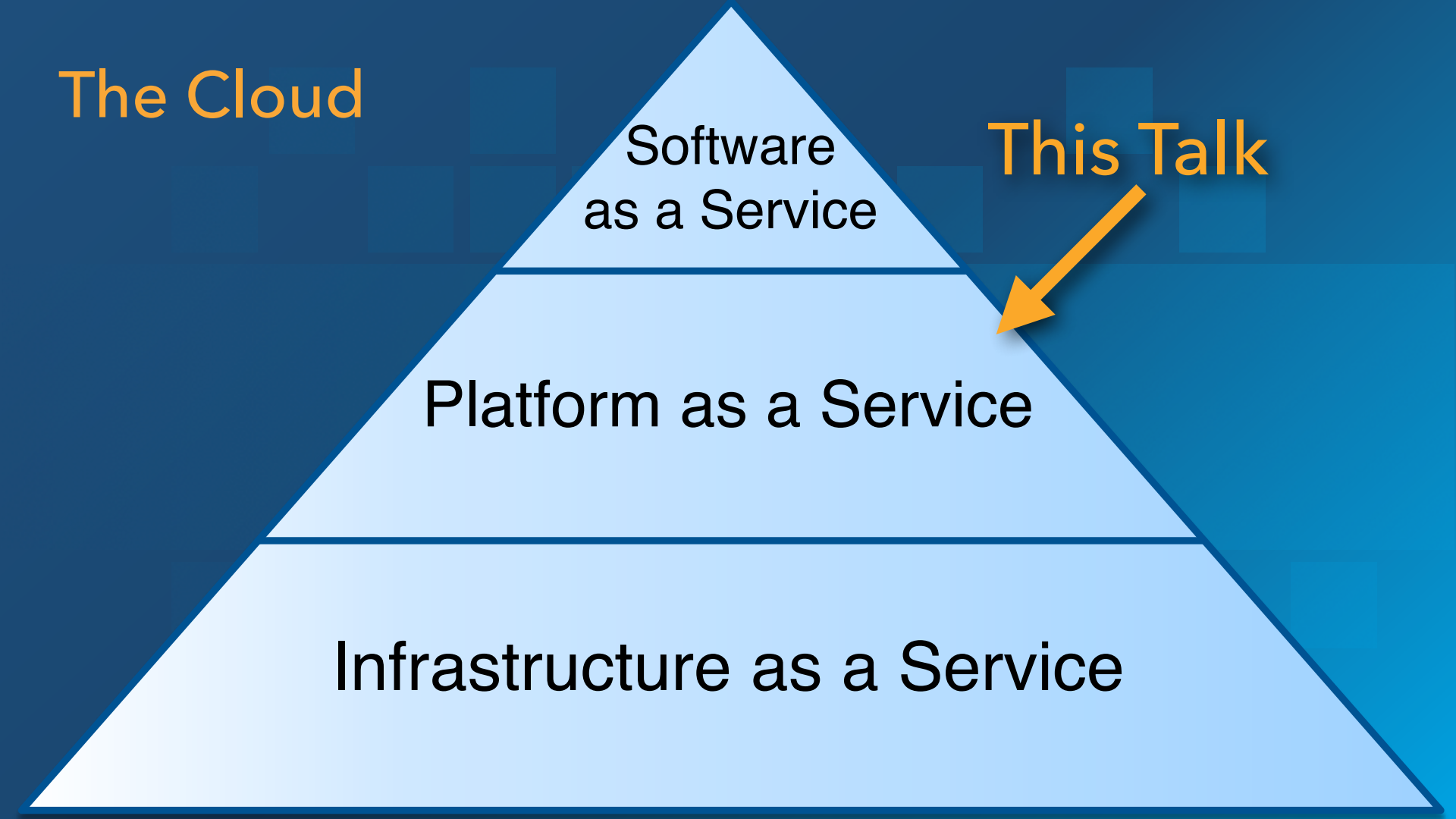
Cloud Application Architecture Patterns

Developing Microservice Applications for PaaS

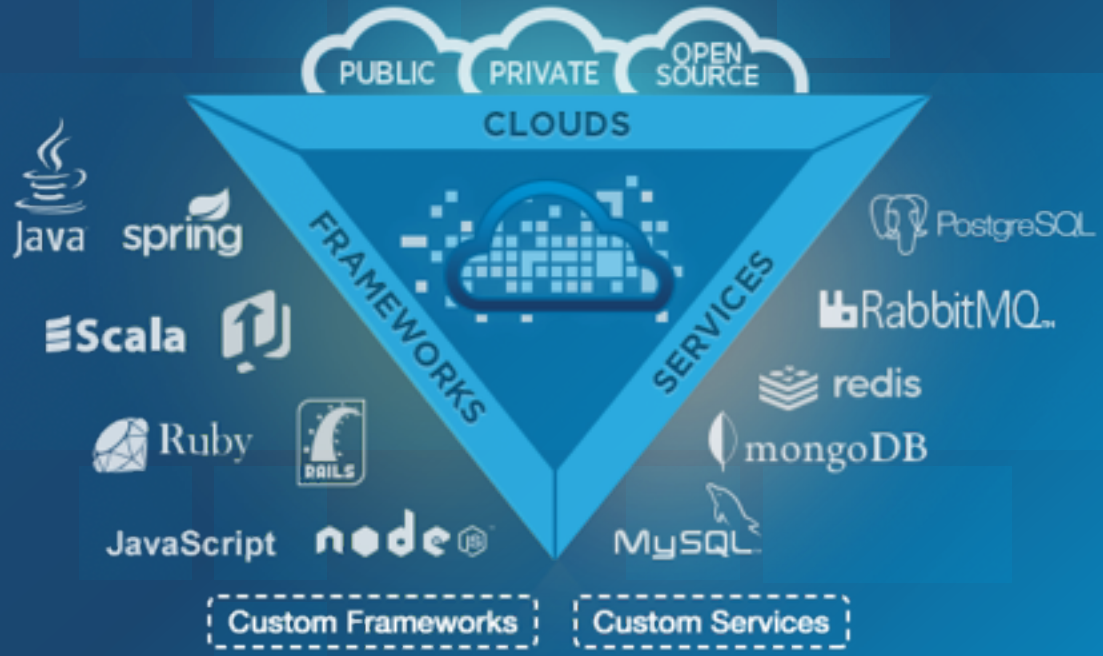
Matt Stine (@mstine)
Cloud Foundry Platform Engineer
matt.stine@gmail.com
http://www.mattstine.com


The Cloud
Software as a Service
This Talk
Platform as a Service
Infrastructure as a Service

PaaS Platforms

- Heroku (http://www.heroku.com)
- OpenShift (https://www.openshift.com)
- Google App Engine (https://appengine.google.com)
- AWS Elastic Beanstalk (https://aws.amazon.com/elasticbeanstalk)
- Microsoft Azure Cloud Services (http://azure.microsoft.com/en-us/services/cloud-services/)
- Cloud Foundry (http://www.cloudfoundry.org)
    - Pivotal Web Services (https://run.pivotal.io)
    - IBM BlueMix (https://ace.ng.bluemix.net)
    - Anynines (http://www.anynines.com)

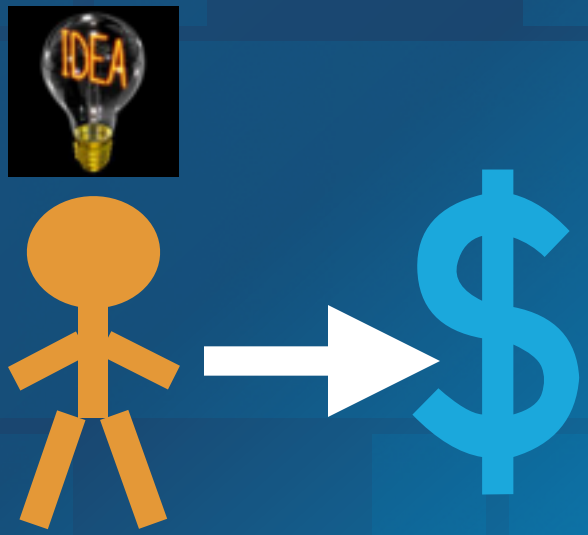
Cloud Foundry PaaS



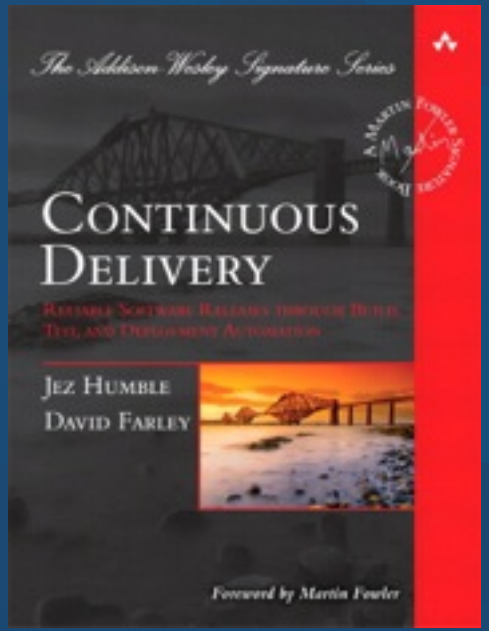
Spring (https://spring.io)

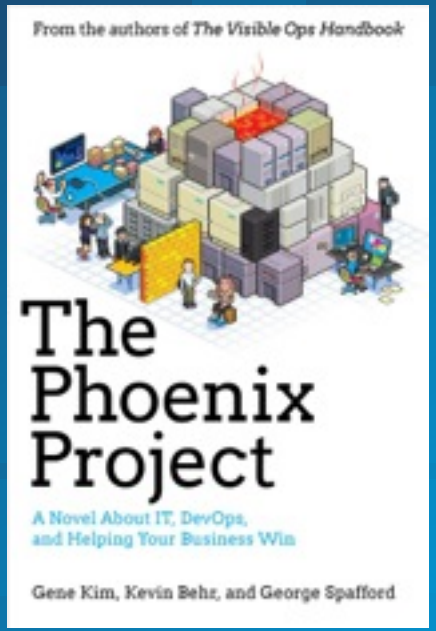
Architecting for Continuous Delivery

Architecting for Continuous Delivery



Continuous Delivery - How?





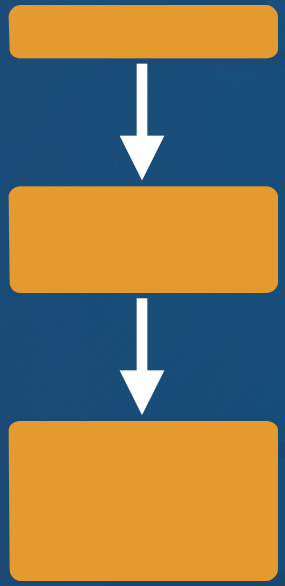
Warner Music: Software Factories

Warner Software Factory Platform

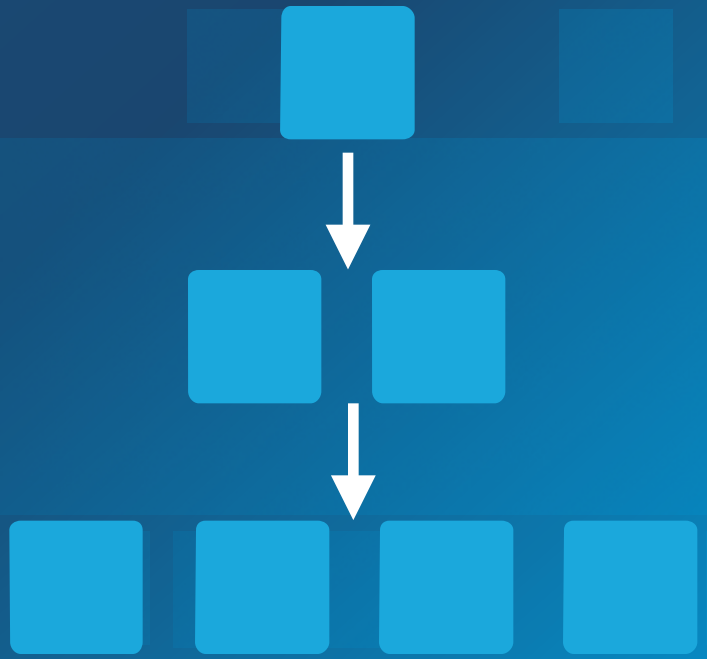
- New applications and major updates
  - **Before**: 6 months, team of 10 developers
  - **After**: 6 weeks, same team
  - **Speed/Agility**: 400% faster on new platform
  - **HR Hard Savings:** $1.1M per application update delivered

Iterative Development

Horizontal Scale



Slow/Expensive



Fast/Cheap

Diversity of Clients

In January 2014, mobile devices accounted for 55% of Internet usage in the United States. Apps made up 47% of Internet traffic and 8% of traffic came from mobile browsers.

http://money.cnn.com/2014/02/28/technology/mobile/mobile-apps-internet/

Continuous Delivery

Software Factories

Feedback → Rapid Iteration

Horizontal Scale

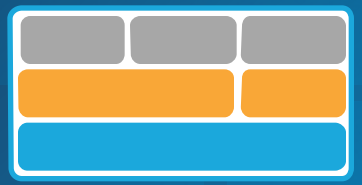
Diversity of Clients

Physical/Virtual

PaaS

Infrastructure

Applications

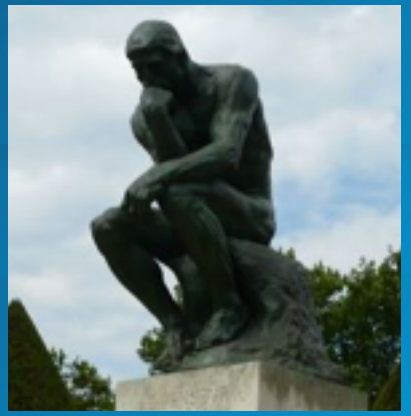


Monoliths

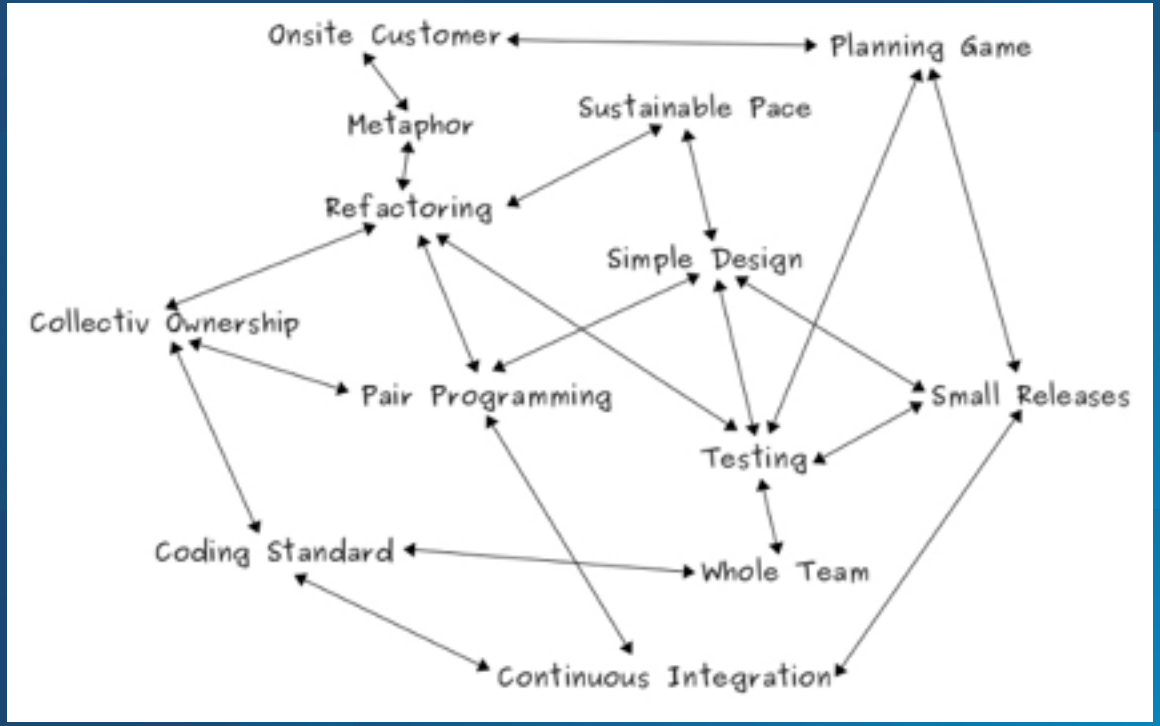
Microservices

New Architectural Constraints

- Platforms like CF and Heroku optimizes for 12 Factor Linux applications
- Microservices: a radical departure from traditional monolithic applications
- In both cases, the enterprise is forced to “think different.”

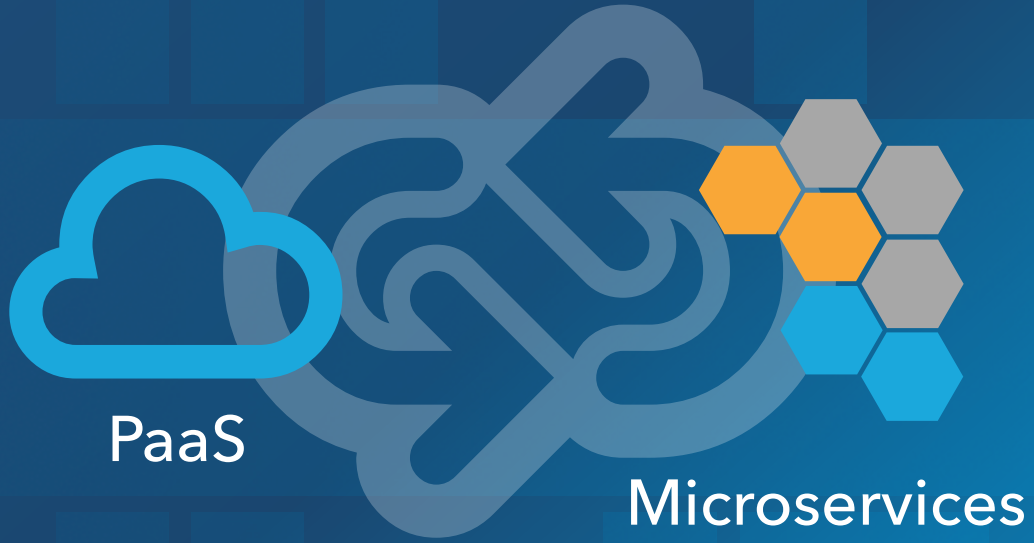


How XP Practices Support Each Other



http://www.kusmin.eu/wiki/index.php/How_XP_Practices_Support_Each_Other

A Symbiotic Relationship...



Patterns

- Microservice
- API Gateway
- Stateless/Shared-Nothing
- Configuration/Service Consumption
- Fault Tolerance

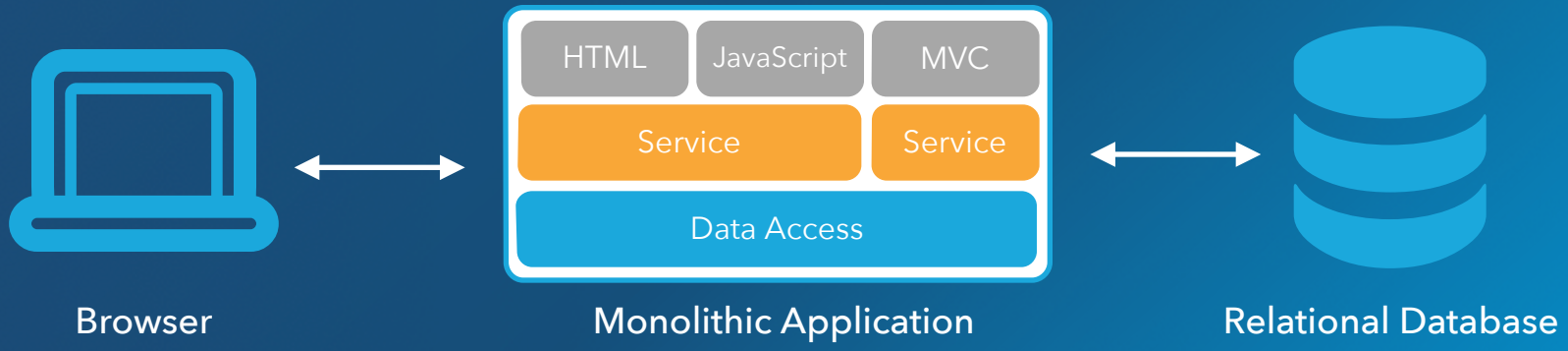
Pattern:

Microservice

Simple vs. Easy

- **Simple**
  - *sim- plex*
  - one fold/braid
  - vs complex
- **Easy**
  - *ease < aise < adjacens*
  - lie near
  - vs hard

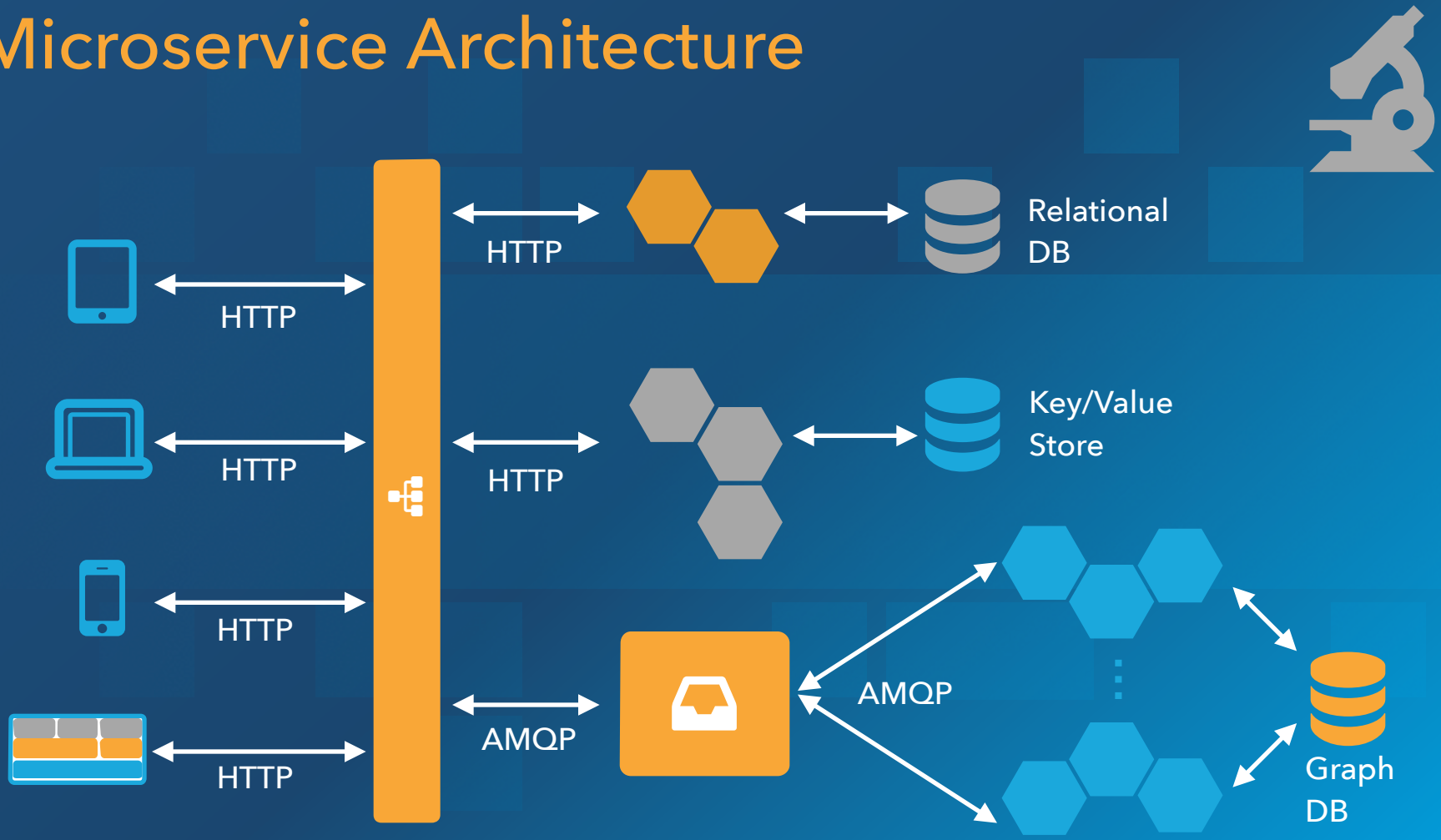
Monolithic Architecture



Monolithic Architectures

- Complex / Easy
- Modularity Dependent Upon Language / Frameworks
- Change Cycles Tightly Coupled / Obstacle to Frequent Deploys
- Inefficient Scaling
- Can Be Intimidating to New Developers
- Obstacle to Scaling Development
- Requires Long-Term Commitment to Technical Stack

Microservice Architecture



Microservice Architectures

- Simple / Challenging
- Modularity Based on Component Services
- Change Cycles Decoupled / Enable Frequent Deploys
- Efficient Scaling
- Individual Components Less Intimidating to New Developers
- Enables Scaling of Development
- Eliminates Long-Term Commitment to Technical Stack

Conway's Law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

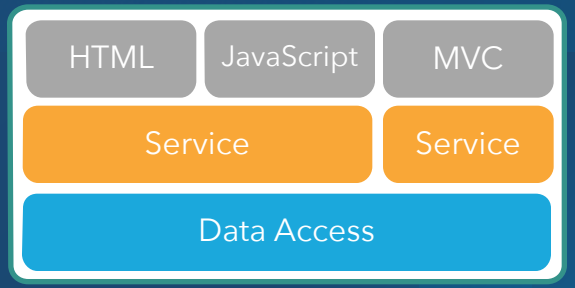
Melvyn Conway, 1967

Organize Around Business Capabilities

Siloed Functional Teams

Cross-functional Teams

Siloed Application Architectures



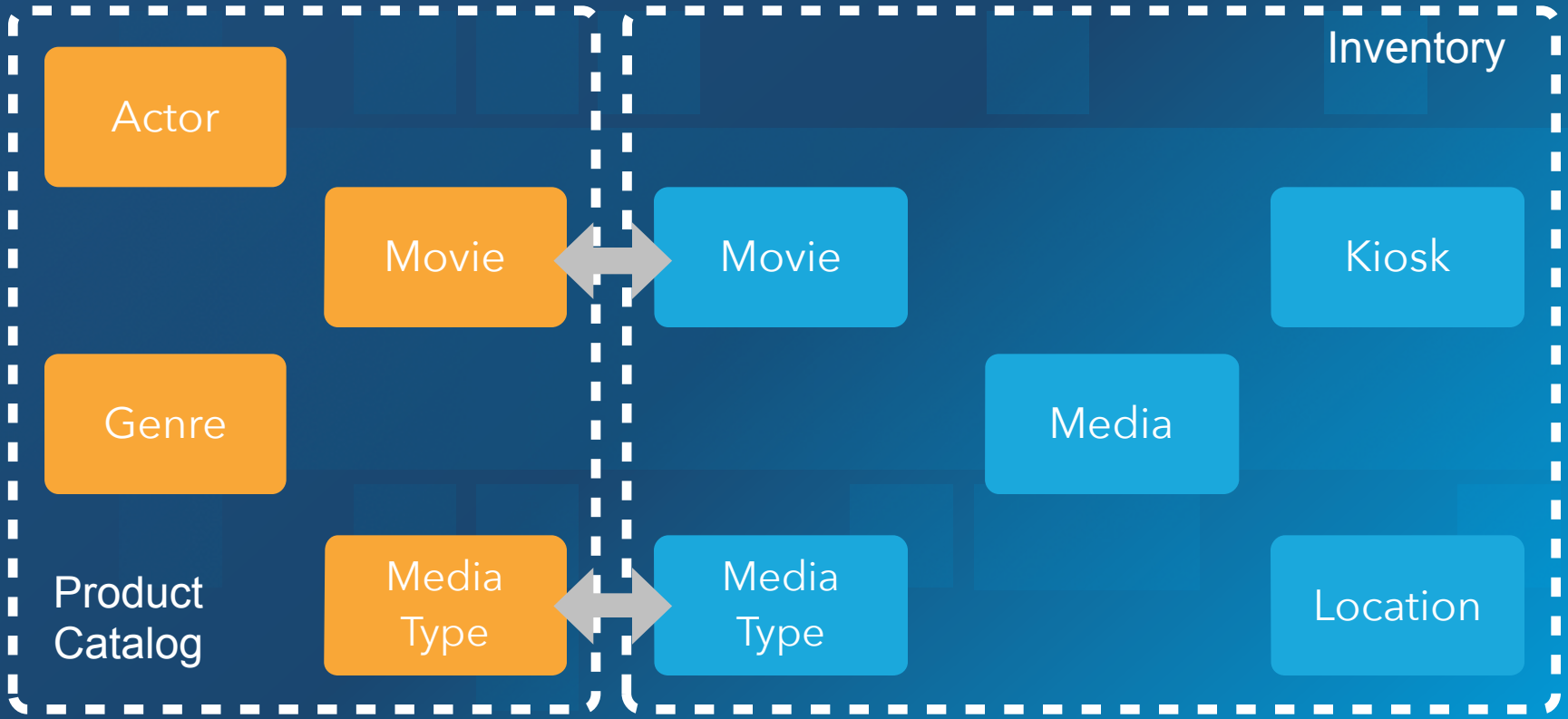
Microservice Architectures

http://martinfowler.com/articles/microservices.html#OrganizedAroundBusinessCapabilities

Partitioning Strategies

- By Noun (e.g. product info service)
- By Verb (e.g. shipping service)
- Single Responsibility Principle (http://programmer.97things.oreilly.com/wiki/index.php/The_Single_Responsibility_Principle)
- Bounded Context (http://martinfowler.com/bliki/BoundedContext.html)

Bounded Contexts



UNIX Pipes and Filters

```
cut -d" " -f1 < access.log | sort | uniq -c | sort -rn | less
```

Choreography over Orchestration

https://www.flickr.com/photos/gabrielsaldana/5896491978

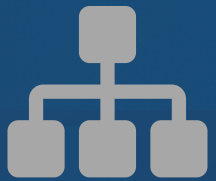
https://www.flickr.com/photos/chrisbrenschmidt/2223763842

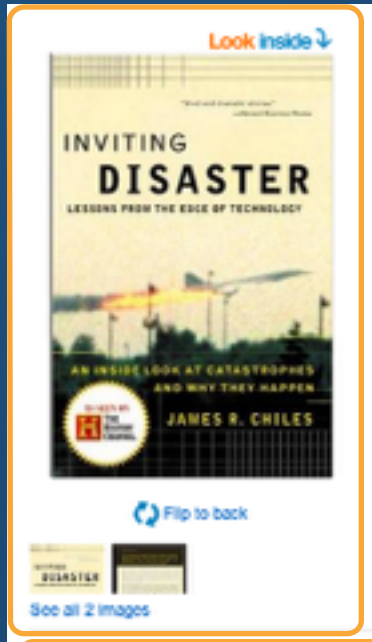
Challenges of Microservices

- Distributed System
- Remote Calls More Expensive Than In-process Calls
- Eventual Consistency
- Features Spanning Multiple Services
- Dependency Management / API Versioning
- Refactoring Module Boundaries

Pattern:

API Gateway





Inviting Disaster: Lessons From the Edge of Technology Paperback

by James R. Chiles (Author)

★★★★½ 98 customer reviews

See all 7 formats and editions

Kindle	Hardcover	Paperback
$10.23	from $0.01	$12.08 Prime
	37 Used from $0.01 14 New from $23.99	50 Used from $0.71 19 New from $6.16 2 Collectible from $9.00

Combining captivating storytelling with eye-opening findings, *Inviting Disaster* delves inside some of history's worst catastrophes in order to show how increasingly "smart" systems leave us wide open to human tragedy.

Weaving a dramatic narrative that explains how breakdowns in these systems result in such disasters as the chain reaction crash of the Air France Concorde to the meltdown at the Chernobyl Nuclear Power Station, Chiles vividly demonstrates how the battle between man and machine may be escalating beyond manageable limits -- and why we all have a stake in its outcome.

Included in this edition is a special introduction providing a behind-the-scenes look at the World Trade Center catastrophe. Combining firsthand accounts of employees' escapes with an in-depth look at the

Read more

Frequently Bought Together

Price for all three: $53.26

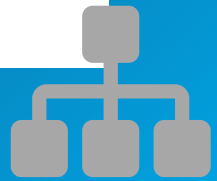
Add all three to Cart | Add all three to Wish List

Show availability and shipping details

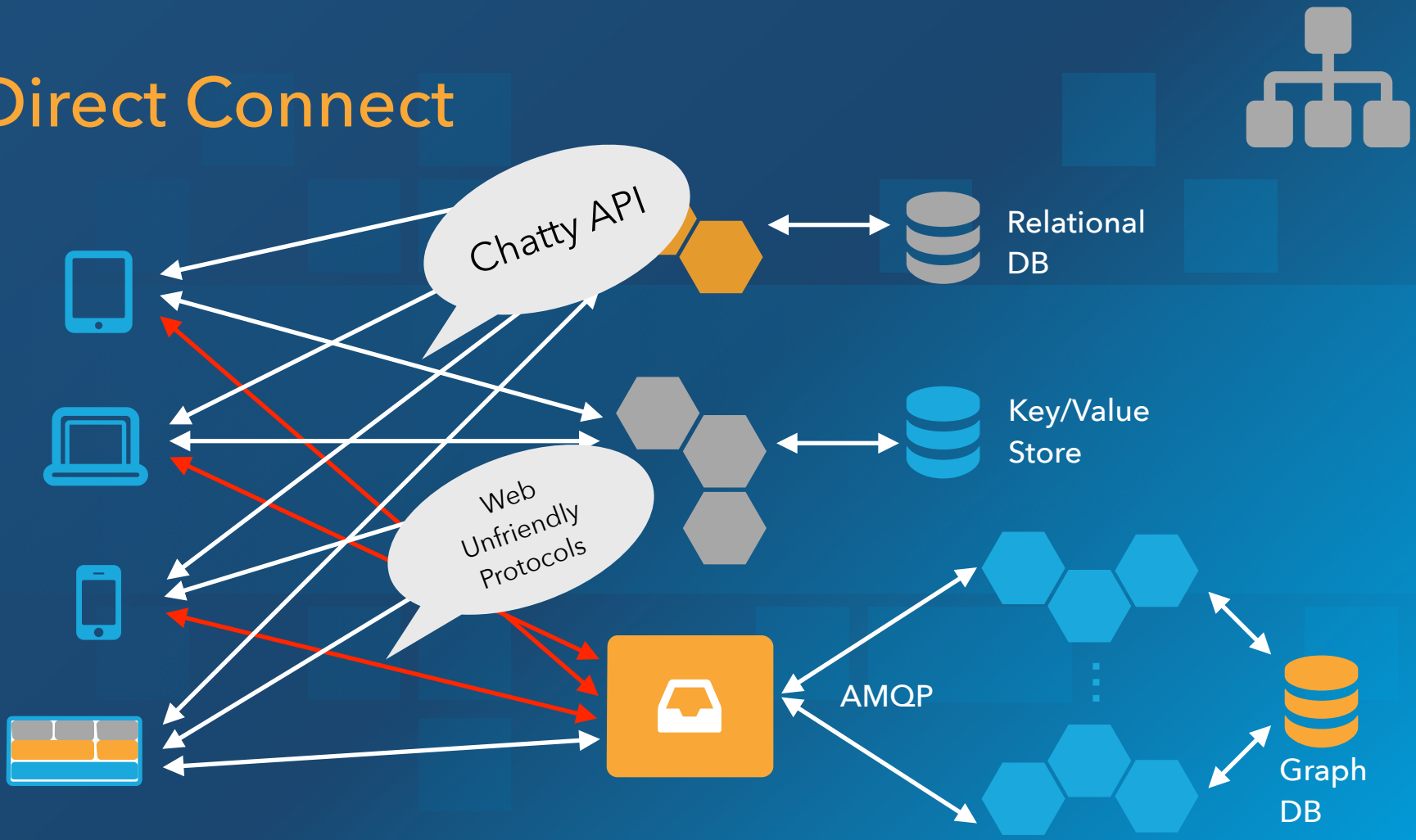
- This item: Inviting Disaster: Lessons From the Edge of Technology by James R. Chiles Paperback $12.08
- The Logic Of Failure: Recognizing And Avoiding Error In Complex Situations by Dietrich Dorner Paperback $12.36
- Normal Accidents: Living with High-Risk Technologies by Charles Perrow Paperback $28.82

How many microservices?

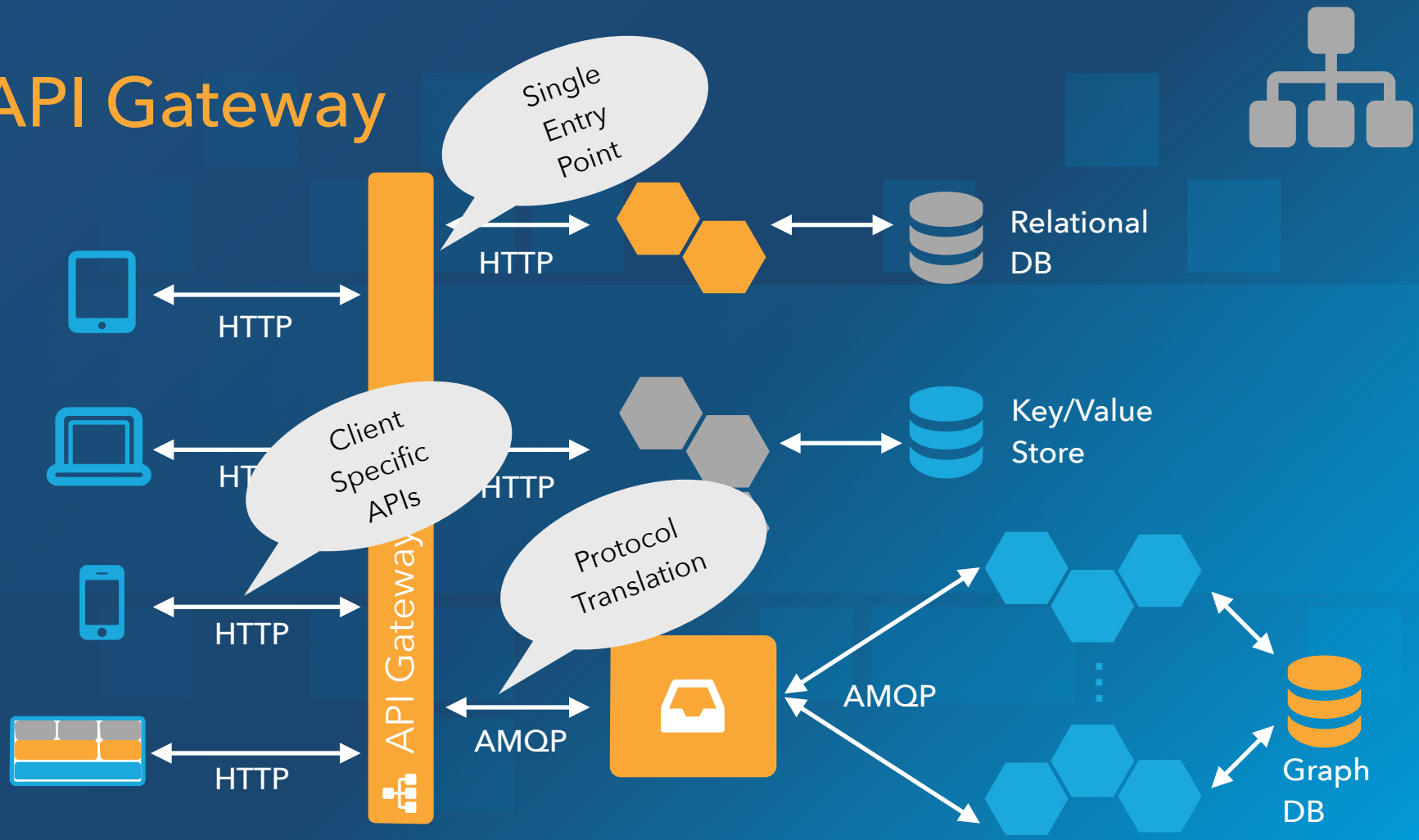
ELEVEN



Direct Connect



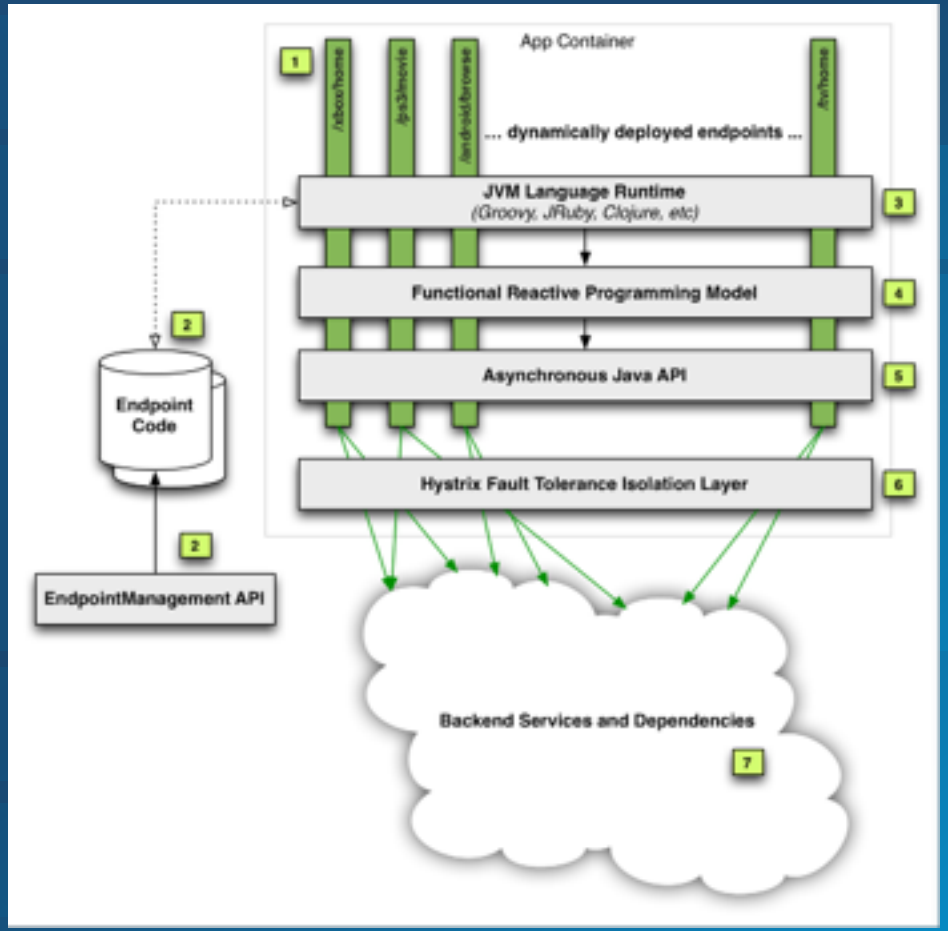
API Gateway


Single Entry Point
HTTP
Relational DB
HTTP
Client Specific APIs
HTTP
Key/Value Store
HTTP
Protocol Translation
API Gateway
AMQP
AMQP
HTTP
Graph DB

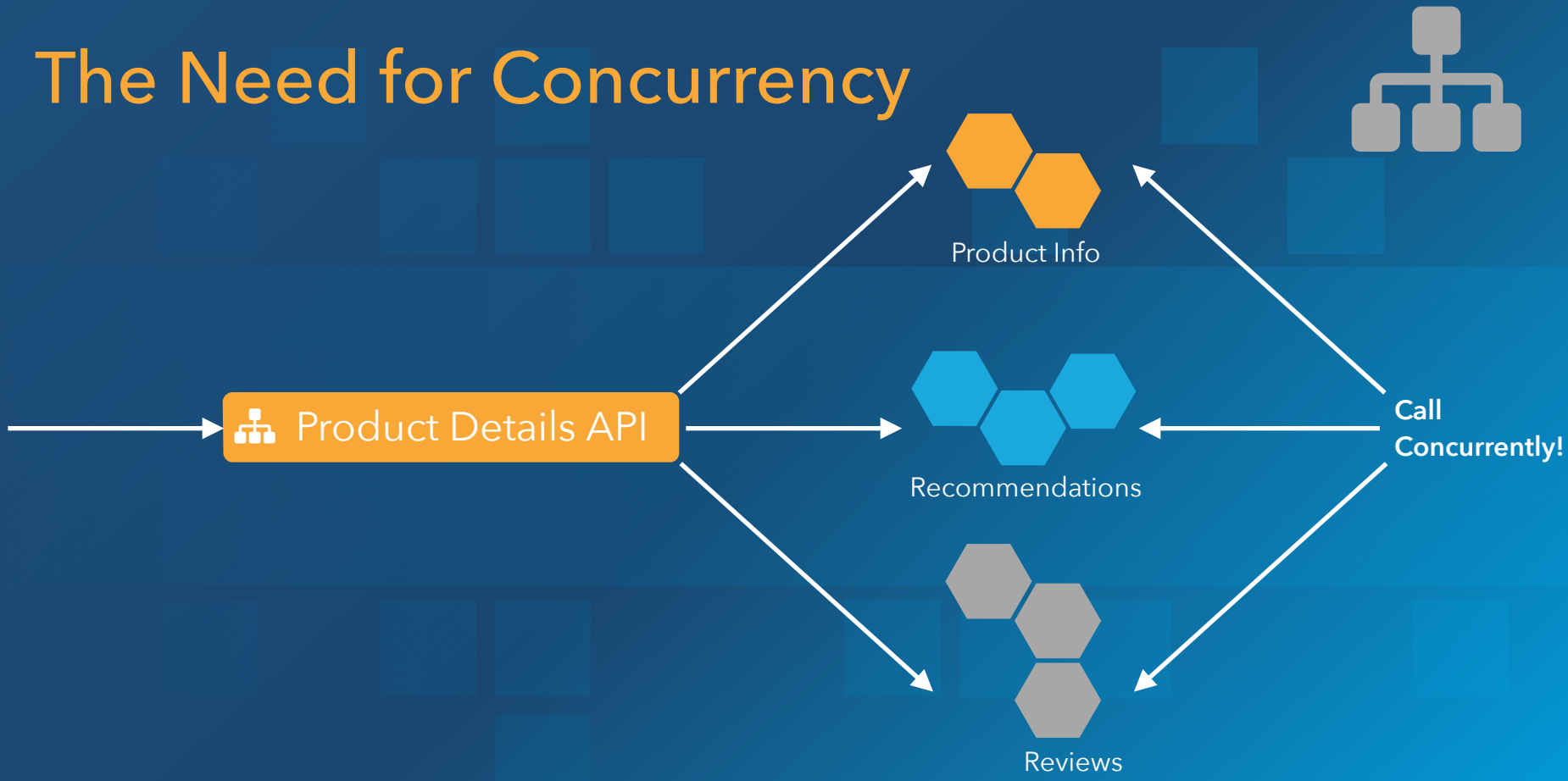
Importance for Mobile

- **Latency**
  - You can’t wait for 2 second or 1 MB responses
- **Round Trips**
  - Too much network == KILL THE BATTERY
- **Aggregation/Transformation**
  - Device-specific responses (manufacturer/device/type/size/etc.)
  - Different Form Factors == Different Use Cases

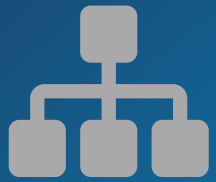
Netflix API Gateway



The Need for Concurrency



Building API Gateways



- **Reactor** (https://spring.io/blog/2013/11/12/it-can-t-just-be-big-data-it-has-to-be-fast-data-reactor-1-0-goes-ga)
- **RxJava** (https://github.com/Netflix/RxJava)
- **Vert.x** (http://vertx.io/)
- **Go** (http://golang.org/)
- **Node.js** (http://nodejs.org/)
- **Pivotal CF Mobile Services** (https://network.pivotal.io/products/p-api-gateway)

Pattern:

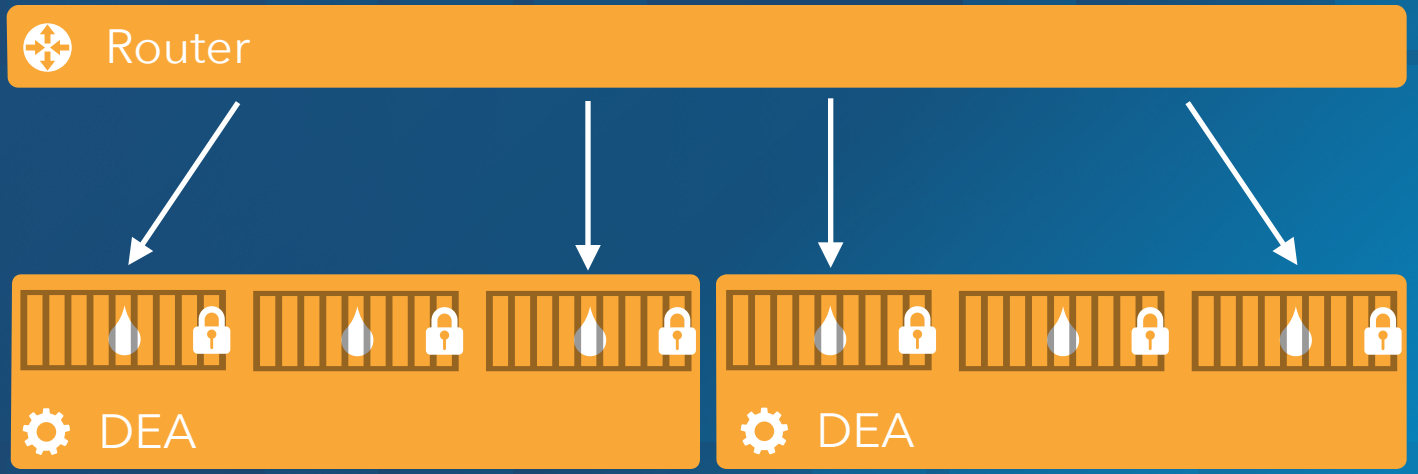
Stateless/Shared-Nothing

Elasticity

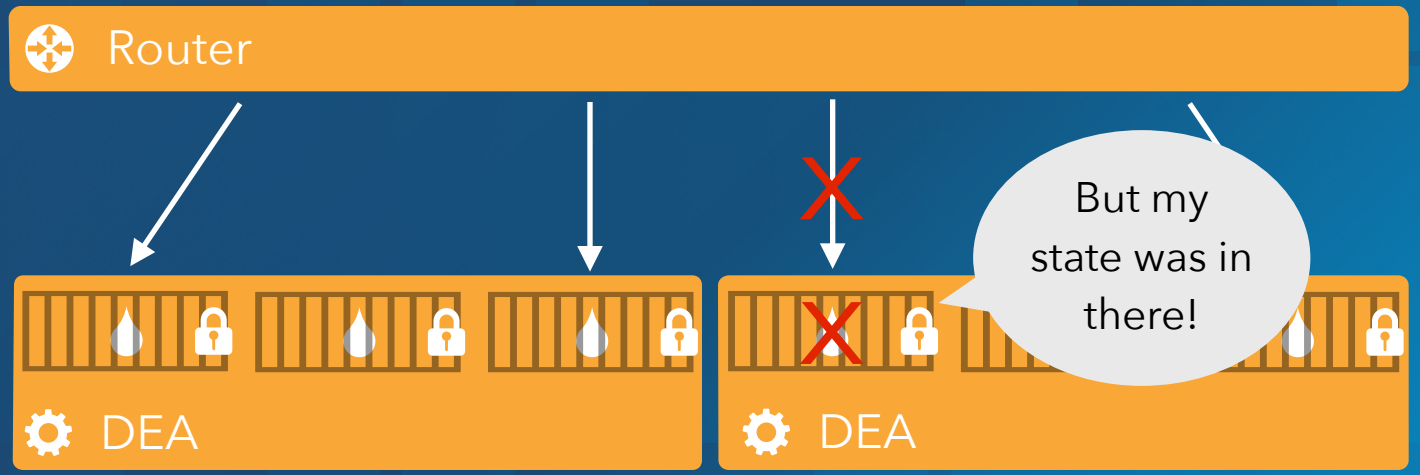
Ephemerality

http://www.flickr.com/photos/smathur/852322080

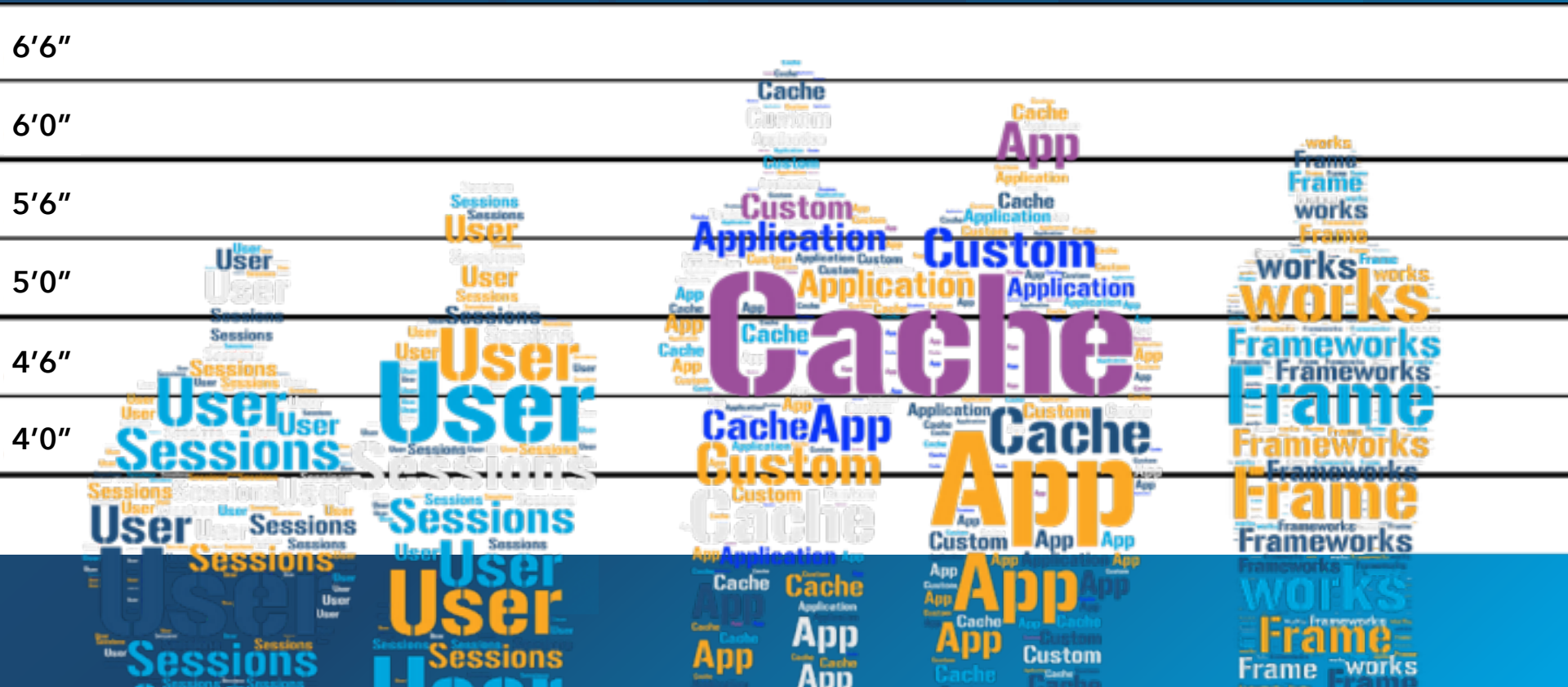
Why is state a problem?



Why is state a problem?



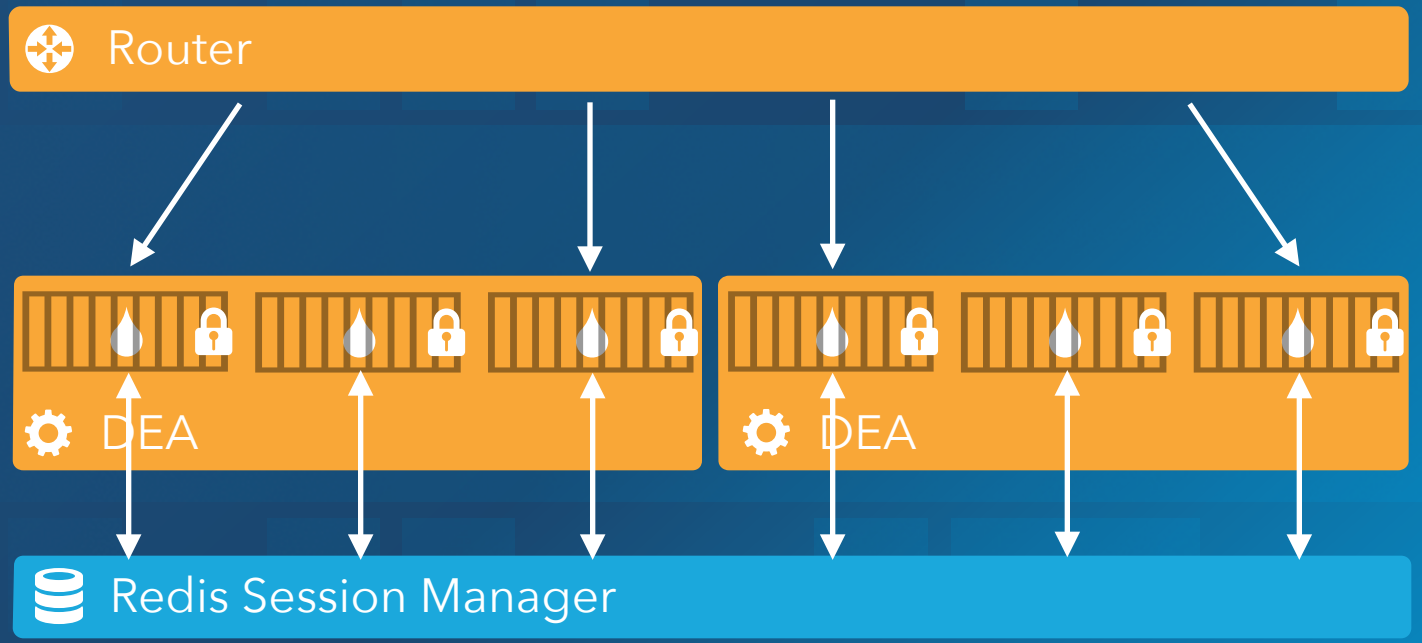
The Usual Suspects



Option #1: Push State to the Client

Cookies, HTML 5, Single Page Applications (SPA)

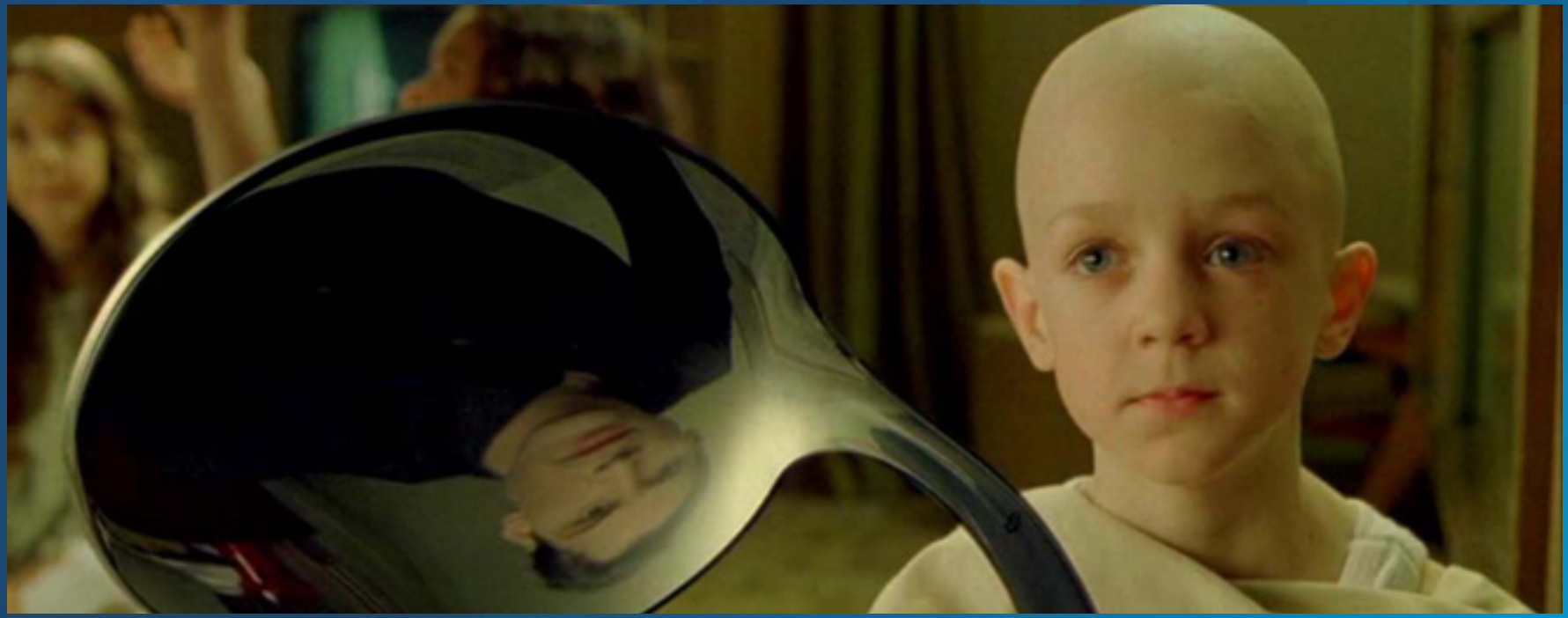
Option #2: External Services



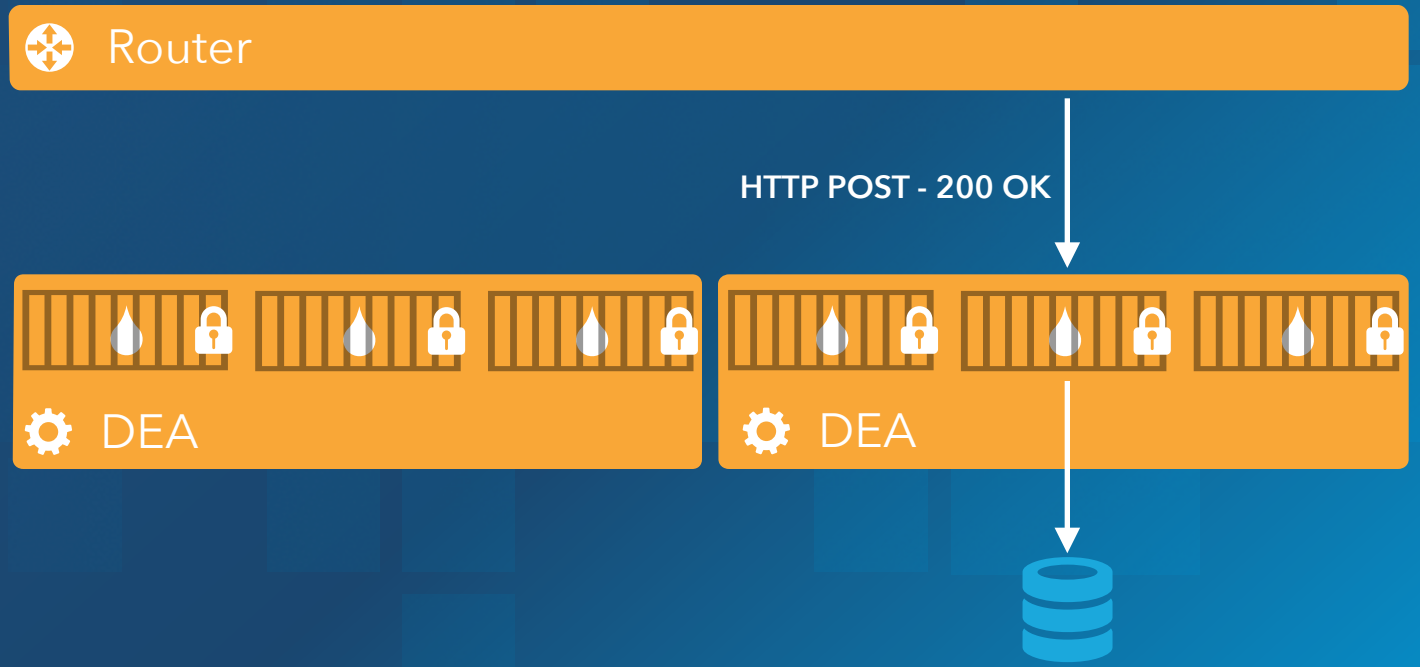
Option #2: External Services

- Spring Session
  - Container-independent session abstraction
  - https://github.com/spring-projects/spring-session
- Redis + CF Java Buildpack
  - Create a CF service containing a name, label, or tag with “session-replication” as a substring.
  - https://github.com/cloudfoundry/java-buildpack/blob/master/docs/container-tomcat.md#session-replication

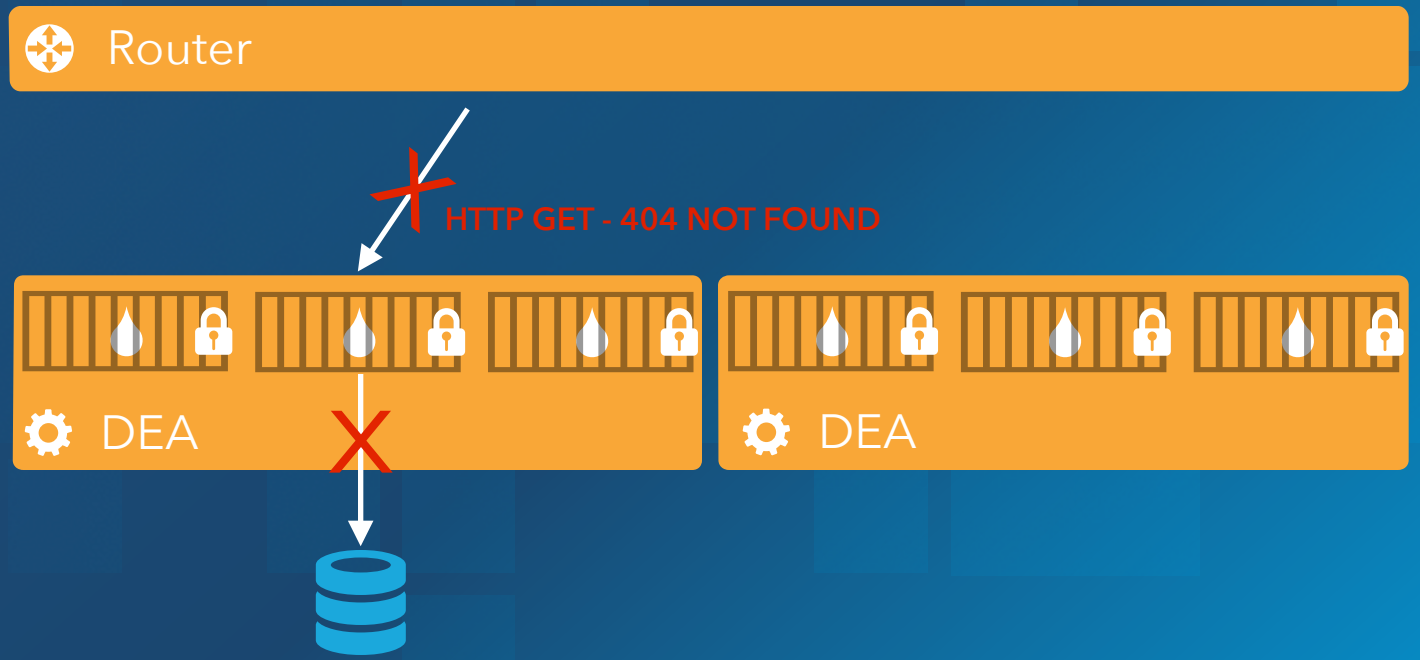
There is no file system...



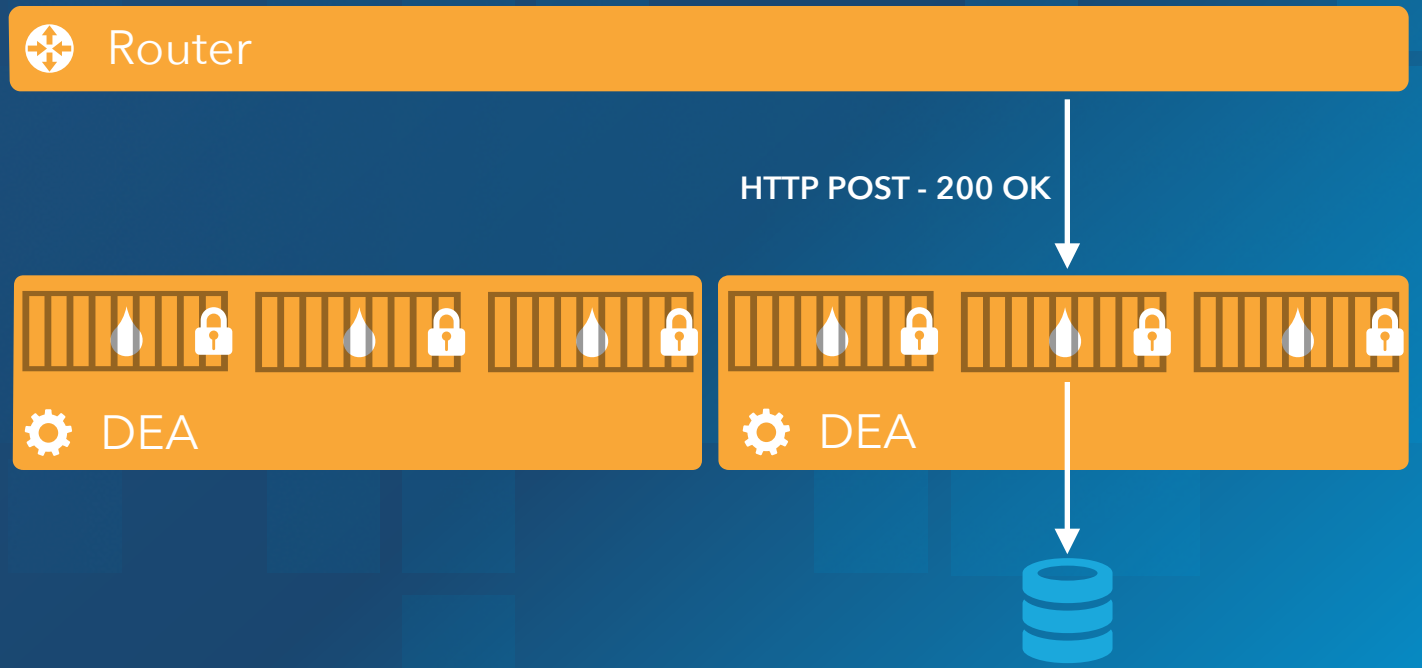
It’s not shared…



It’s not shared…



It's not persistent...

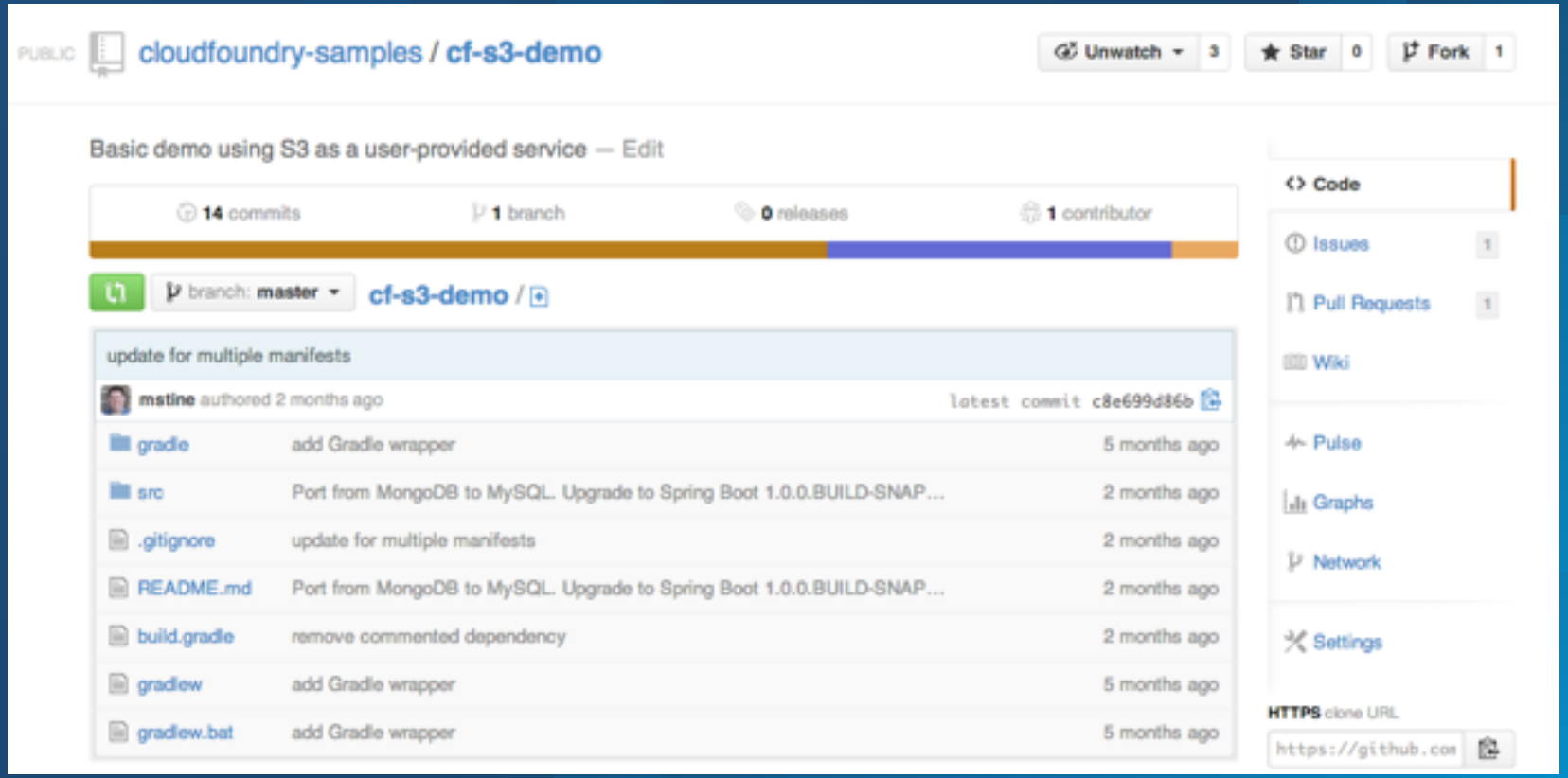


It’s not persistent...

It’s not persistent...

Use a persistent store!

- Amazon S3 (http://aws.amazon.com/s3)
- Google Cloud Storage (https://cloud.google.com/products/cloud-storage)
- OpenStack Swift (http://swift.openstack.org)
- Hadoop HDFS (http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html)
- Riak CS for Pivotal CF (https://network.pivotal.io/products/p-riakcs)



https://github.com/cloudfoundry-samples/cf-s3-demo

```
@Configuration
@Profile("cloud")
public class CloudConfig extends AbstractCloudConfig {

   @Bean
   public DataSource dataSource() {
      // Default pool size to 4 connections to support ClearDB Spark (free)
      PooledServiceConnectorConfig.PoolConfig poolConfig =
            new PooledServiceConnectorConfig.PoolConfig(4, 200);

      DataSourceConfig config = new DataSourceConfig(poolConfig, new DataSourceConfig.
                        ConnectionConfig(""));

      return connectionFactory().dataSource(config);
   }

   @Bean
   public S3 s3() {
      return connectionFactory().service(S3.class);
   }

}
```

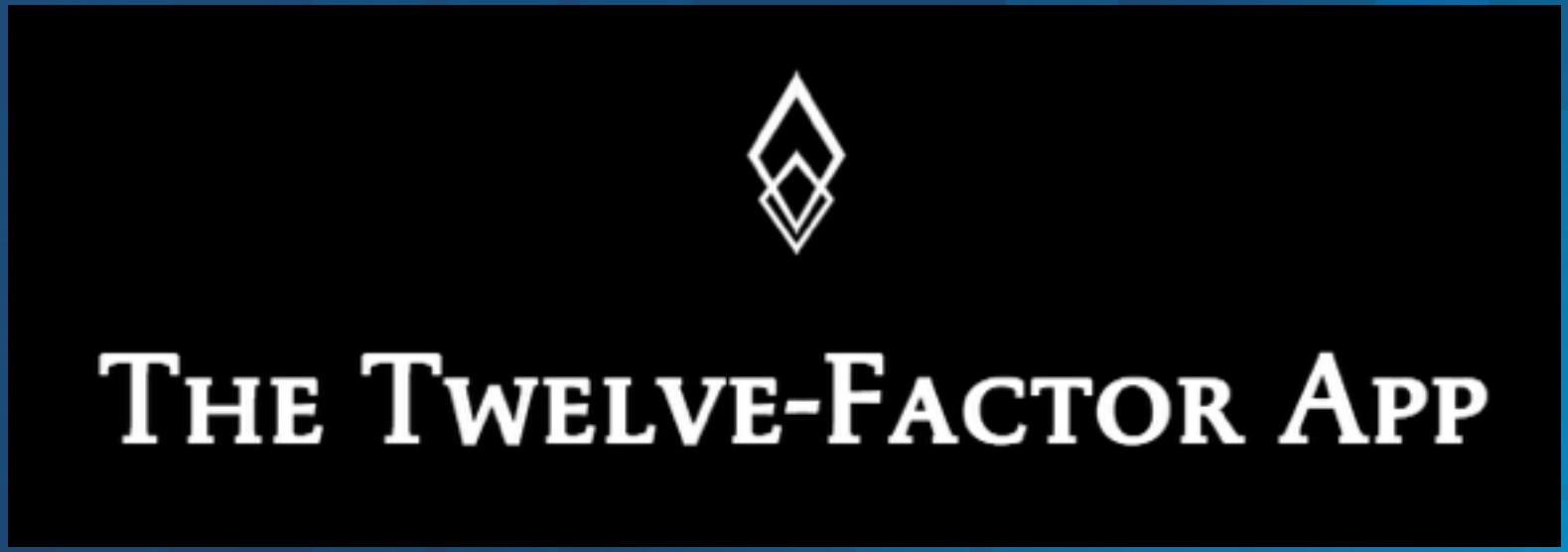
```
public class S3 {

    private AmazonS3 amazonS3;
    private String bucket;

    public S3(AmazonS3 amazonS3, String bucket) {
        this.amazonS3 = amazonS3;
        this.bucket = bucket;
    }

    public S3File createS3FileObject(String id, String name, File file) {
        return new S3File(id, bucket, name, file);
    }

    public void put(S3File file) {
        PutObjectRequest putObjectRequest = new PutObjectRequest(bucket, file.getActualFileName(),
                                        file.getFile());
        putObjectRequest.withCannedAcl(CannedAccessControlList.PublicRead);
        amazonS3.putObject(putObjectRequest);
    }

    public void delete(S3File file) {
        amazonS3.deleteObject(bucket, file.getActualFileName());
    }

}
```

```
@RequestMapping(value = "/upload", method = RequestMethod.POST)
    public String handleFileUpload(@RequestParam("file") MultipartFile file) {

        String id = UUID.randomUUID().toString();
        File uploadedFile = new File(file.getOriginalFilename());

        try {
            byte[] bytes = file.getBytes();
            BufferedOutputStream stream = new BufferedOutputStream(new FileOutputStream(uploadedFile)
                                        );
            stream.write(bytes);
            stream.close();
        } catch (IOException e) {
            throw new RuntimeException("Failed to upload file!", e);
        }

        S3File s3File = s3.createS3FileObject(id, file.getOriginalFilename(), uploadedFile);

        s3.put(s3File);
        log.info(s3File.getName() + " put to S3.");
        repository.save(s3File);
        log.info(s3File.getName() + " record saved to MySQL.");
        uploadedFile.delete();
        log.info(s3File.getFile().getAbsolutePath() + " is deleted.");

        return “redirect:/";
}
```

```
@RequestMapping(value = "/delete/{id}", method = RequestMethod.GET)
    public String deleteFile(@PathVariable String id) {

        S3File s3File = repository.findOne(id);

        repository.delete(s3File);
        log.info(s3File.getId() + " deleted from MySQL.");
        s3.delete(s3File);
        log.info(s3File.getActualFileName() + " deleted from S3 bucket.");

        return "redirect:/";
}
```

Pattern:

Configuration/Service Consumption



http://12factor.net/

What is configuration?

- Resource handles to databases and other backing services
- Credentials to external sources (e.g. S3, Twitter, ...)
- Per-deploy values (e.g. canonical hostname for deploy)
- ANYTHING that's likely to vary between deploys (dev, test, stage, prod)

Where NOT to store it:

- In the **CODE** (Obvious)
- In **PROPERTIES FILES** (That's code...)
- In the **BUILD** (ONE build, MANY deploys)
- In the **APP SERVER** (e.g. JNDI datasources)

Store it in the ENVIRONMENT!

```
TMPDIR=/home/vcap/tmp
VCAP_APP_PORT=61863
USER=vcap
VCAP_APPLICATION={"instance_id":"b3e92a6fc443436888a525d100c91a12",
                  "instance_index":0,"host":"0.0.0.0","port":61863,"started_at":
                  "2013-12-04 01:52:01 +0000","started_at_timestamp":1386121921,
                  "start":"2013-12-04 01:52:01+0000","state_timestamp":1386121921,
                  "limits":{"mem":512,"disk":1024,"fds":16384},
                  "application_version":"09c9bfe9-c14e-4fcb-8ad8-9fcd4b854893",
                  "application_name":"cf-s3-demo","application_uris":["cf-s3-demo.
                  cfapps.io"],"version":"09c9bfe9-c14e-4fcb-8ad8-9fcd4b854893",
                  "name":"cf-s3-demo","uris":["cf-s3-demo.cfapps.io"],"users":
                  null}
PATH=/bin:/usr/bin
PWD=/home/vcap
VCAP_SERVICES={"mongolab-n/a":[{"name":"mongo-cf-s3","label":"mongolab-n/a",
               "tags":["mongodb","document"],"plan":"sandbox","credentials":{
               "uri":"mongodb://user:****@ds053708.mongolab.com:
               53708/CloudFoundry_shageik2_iijfo9ve"}}],"user-provided":[{"name":
               "s3-bucket-service","label":"user-provided","tags":[],
               "credentials":{"awsAccessKey":"**********","awsSecretKey":
               "**********","bucket":"cf-s3-demo"},"syslog_drain_url":""}]}
SHLVL=1
HOME=/home/vcap/app
PORT=61863
VCAP_APP_HOST=0.0.0.0
DATABASE_URL=
MEMORY_LIMIT=512m
_=/usr/bin/env
```

Cloud Foundry Environment

When am I done?

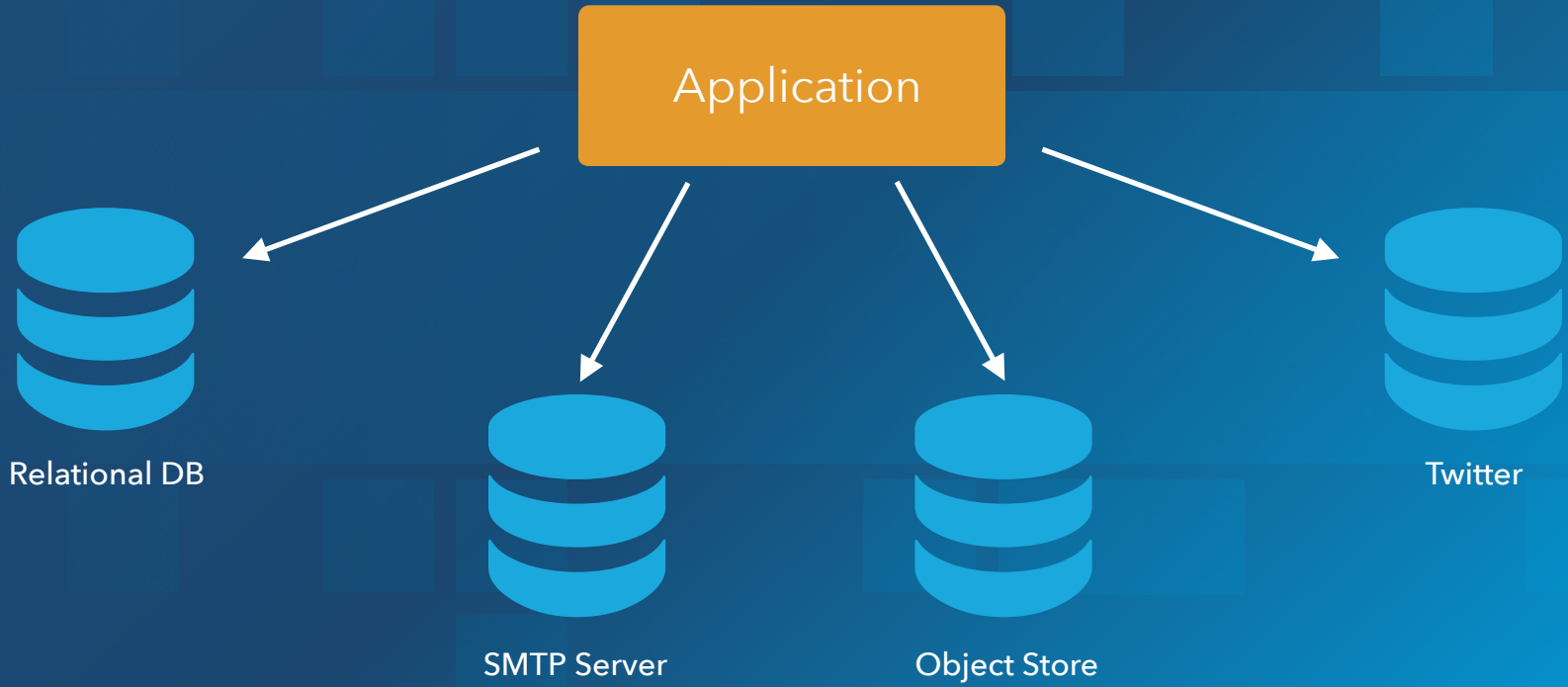
When "...the codebase could be made open source at any moment, without compromising any credentials."

http://12factor.net/config

Why environment variables?

- Easy to change
- Little chance of being “checked in” to VCS
- Language/OS-agnostic standard

Backing Services



Cloud Foundry Environment

```
TMPDIR=/home/vcap/tmp
VCAP_APP_PORT=61863
USER=vcap
VCAP_APPLICATION={"instance_id":"b3e92a6fc443436888a525d100c91a12",
                  "instance_index":0,"host":"0.0.0.0","port":61863,"started_at":
                  "2013-12-04 01:52:01 +0000","started_at_timestamp":1386121921,
                  "start":"2013-12-04 01:52:01+0000","state_timestamp":1386121921,
                  "limits":{"mem":512,"disk":1024,"fds":16384},
                  "application_version":"09c9bfe9-c14e-4fcb-8ad8-9fcd4b854893",
                  "application_name":"cf-s3-demo","application_uris":["cf-s3-demo.
                  cfapps.io"],"version":"09c9bfe9-c14e-4fcb-8ad8-9fcd4b854893",
                  "name":"cf-s3-demo","uris":["cf-s3-demo.cfapps.io"],"users":
                  null}
PATH=/bin:/usr/bin
PWD=/home/vcap
VCAP_SERVICES={"mongolab-n/a":[{"name":"mongo-cf-s3","label":"mongolab-n/a",
               "tags":["mongodb","document"],"plan":"sandbox","credentials":{
               "uri":"mongodb://user:****@ds053708.mongolab.com:
               53708/CloudFoundry_shageik2_iijfo9ve"}}],"user-provided":[{"name":
               "s3-bucket-service","label":"user-provided","tags":[],
               "credentials":{"awsAccessKey":"**********","awsSecretKey":
               "**********","bucket":"cf-s3-demo"},"syslog_drain_url":""}]}
SHLVL=1
HOME=/home/vcap/app
PORT=61863
VCAP_APP_HOST=0.0.0.0
DATABASE_URL=
MEMORY_LIMIT=512m
_=/usr/bin/env
```

spring

DOCS GUIDES PROJECTS BLOG FORUM

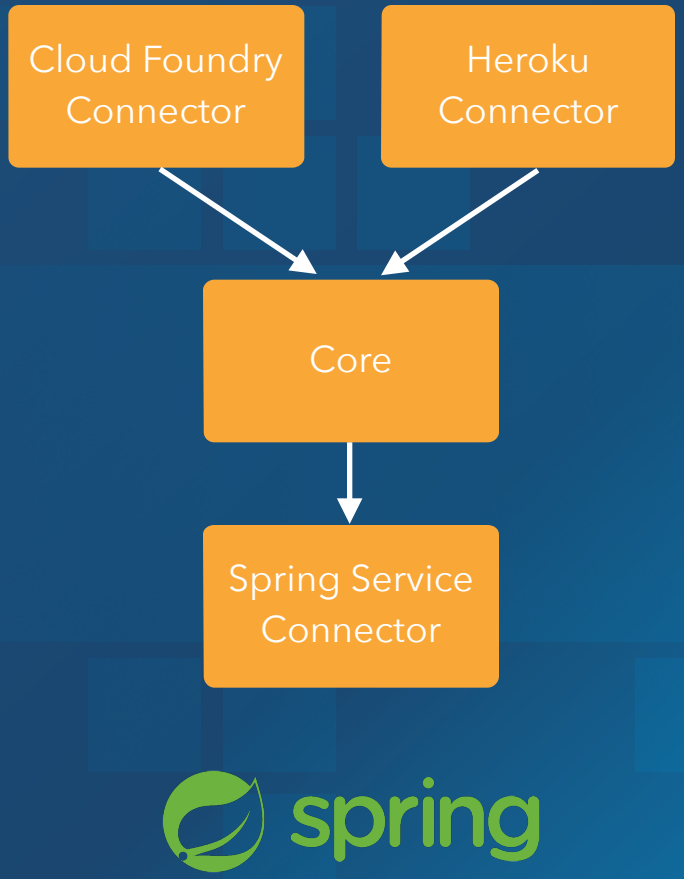
PROJECTS

Spring Cloud Connectors

Spring Cloud Connectors simplifies connecting to services and gaining operating environment awareness in cloud platforms like Cloud Foundry and Heroku. Special support for Spring application through Java and XML config makes it trivial for apps to connect to cloud services. Designed for extensibility, you can use one of the existing cloud connectors (Cloud Foundry and Heroku) or write one for your cloud platform. While supporting commonly used services (relational databases, MongoDB, Redis, Rabbit) out of the box, it allows extending it to your own services. Neither of these require modifying Spring Cloud itself; all you need to do is add jars for your extensions to your classpath.

QUICK START

http://cloud.spring.io/spring-cloud-connectors/



Spring Cloud Connectors Modules

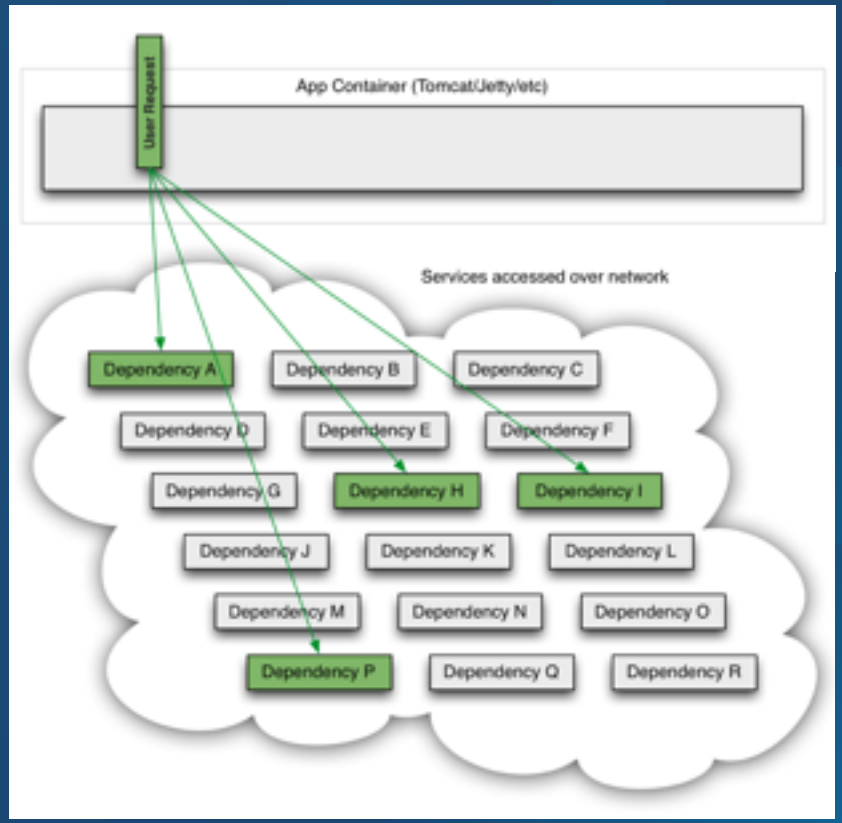
Spring Service Beans

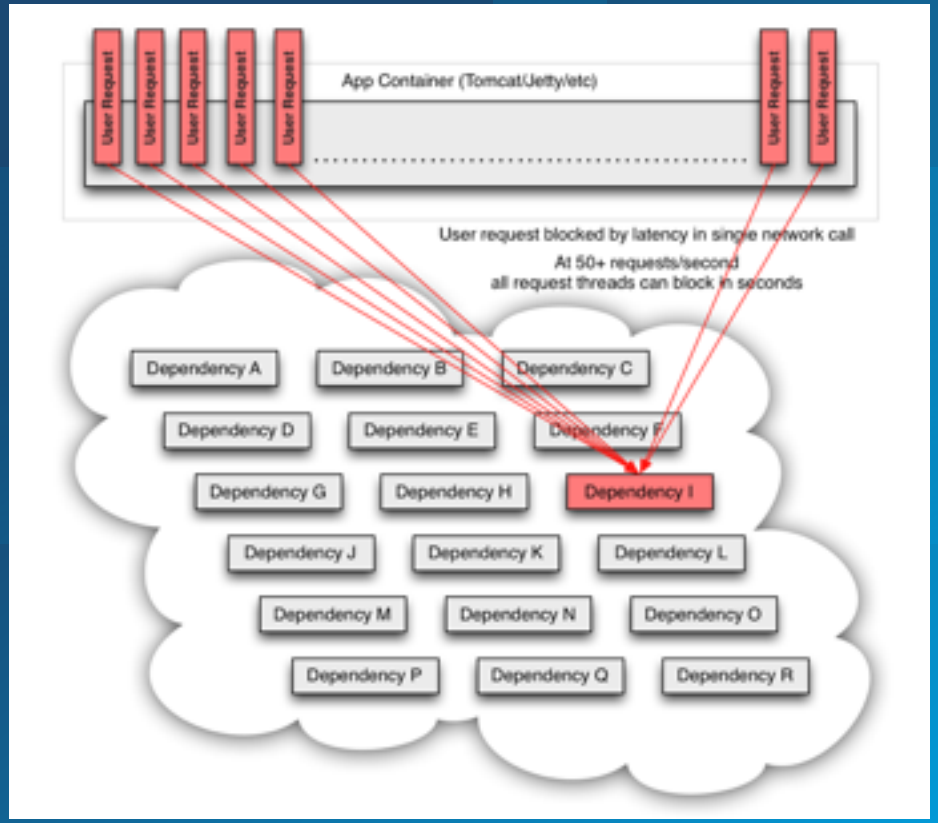
```
public class CloudConfig extends AbstractCloudConfig {
    @Bean
    public ConnectionFactory rabbitConnectionFactory() {
      return connectionFactory.rabbitConnectionFactory();
    }

    @Bean
    public DataSource dataSource() {
        return connectionFactory().dataSource();
    }

    @Bean
    public S3 s3() {
        return connectionFactory().service(S3.class);
    }
}
```

Pattern:

Fault Tolerance

Fault Tolerance at Netflix





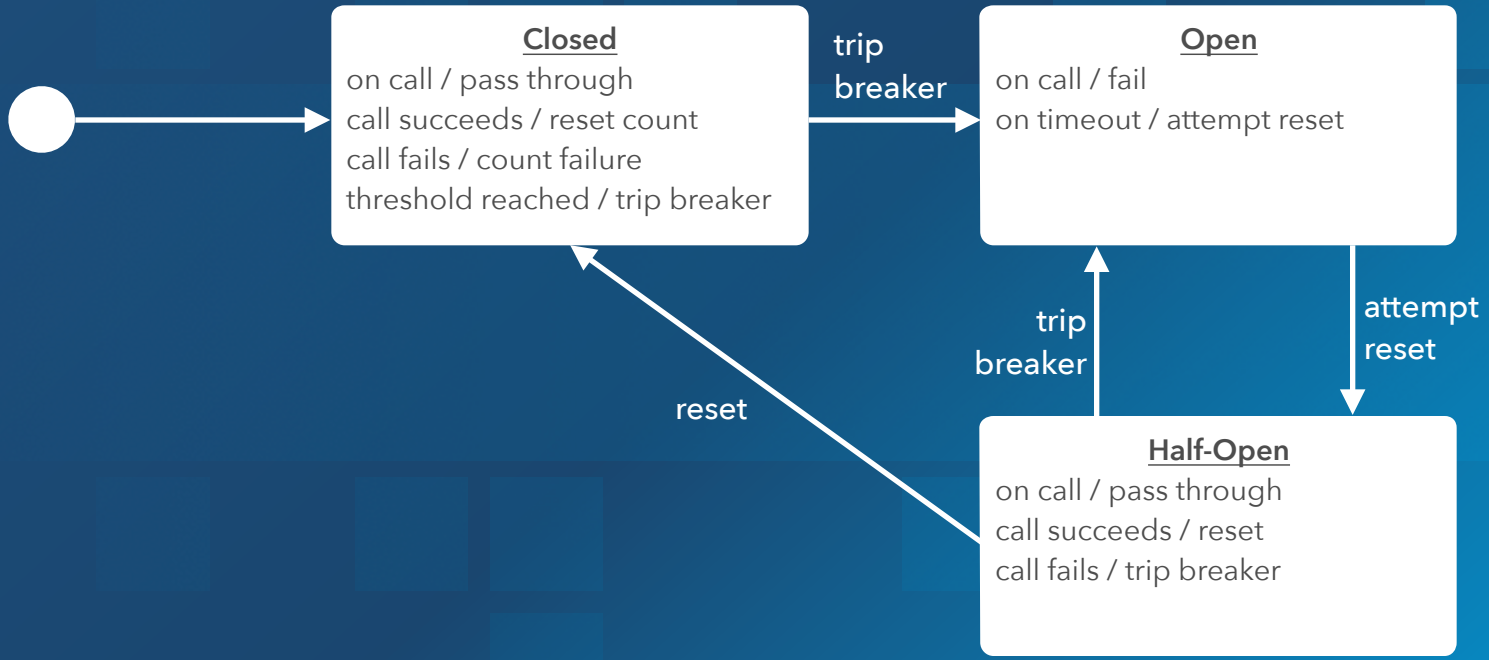
http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

Without taking steps to ensure fault tolerance, 30 dependencies each with 99.99% uptime would result in 2+ hours downtime/month (99.99%^{30} = 99.7% uptime = 2+ hours downtime in a month).

http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html

Circuit Breaker

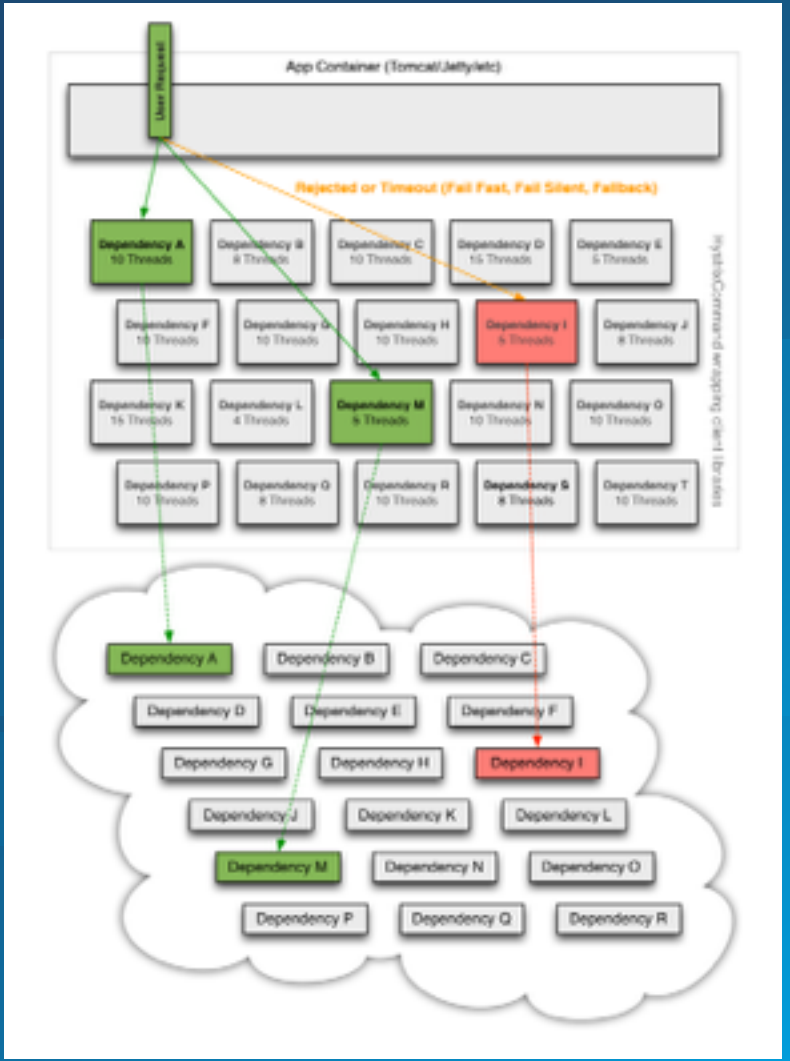
Circuit Breaker State Machine



https://github.com/Netflix/Hystrix

Hystrix

- Latency and Fault Tolerance
- Realtime Operations
- Concurrency



Hello Hystrix!

```
public class CommandHelloWorld extends HystrixCommand<String> {

    private final String name;

    public CommandHelloWorld(String name) {
        super(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"));
        this.name = name;
    }

    @Override
    protected String run() {
        return "Hello " + name + "!";
    }
}
```

Consuming Hystrix Commands

```
String s = new CommandHelloWorld("Bob").execute();

Future<String> s = new CommandHelloWorld("Bob").queue();

Observable<String> s = new CommandHelloWorld("Bob").observe();

s.subscribe(new Action1<String>() {

    @Override
    public void call(String s) {
         // value emitted here
    }

});
```

Fail Fast

```
public class CommandThatFailsFast extends HystrixCommand<String> {

    private final boolean throwException;

    public CommandThatFailsFast(boolean throwException) {
        super(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"));
        this.throwException = throwException;
    }

    @Override
    protected String run() {
        if (throwException) {
            throw new RuntimeException("failure from CommandThatFailsFast");
        } else {
            return "success";
        }
    }
```

Fail Silent

```
public class CommandThatFailsSilently extends HystrixCommand<String> {

    private final boolean throwException;

    public CommandThatFailsSilently(boolean throwException) {
        super(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"));
        this.throwException = throwException;
    }

    @Override
    protected String run() {
        if (throwException) {
            throw new RuntimeException("failure from CommandThatFailsFast");
        } else {
            return "success";
        }
    }

    @Override
    protected String getFallback() {
        return null;
    }
}
```

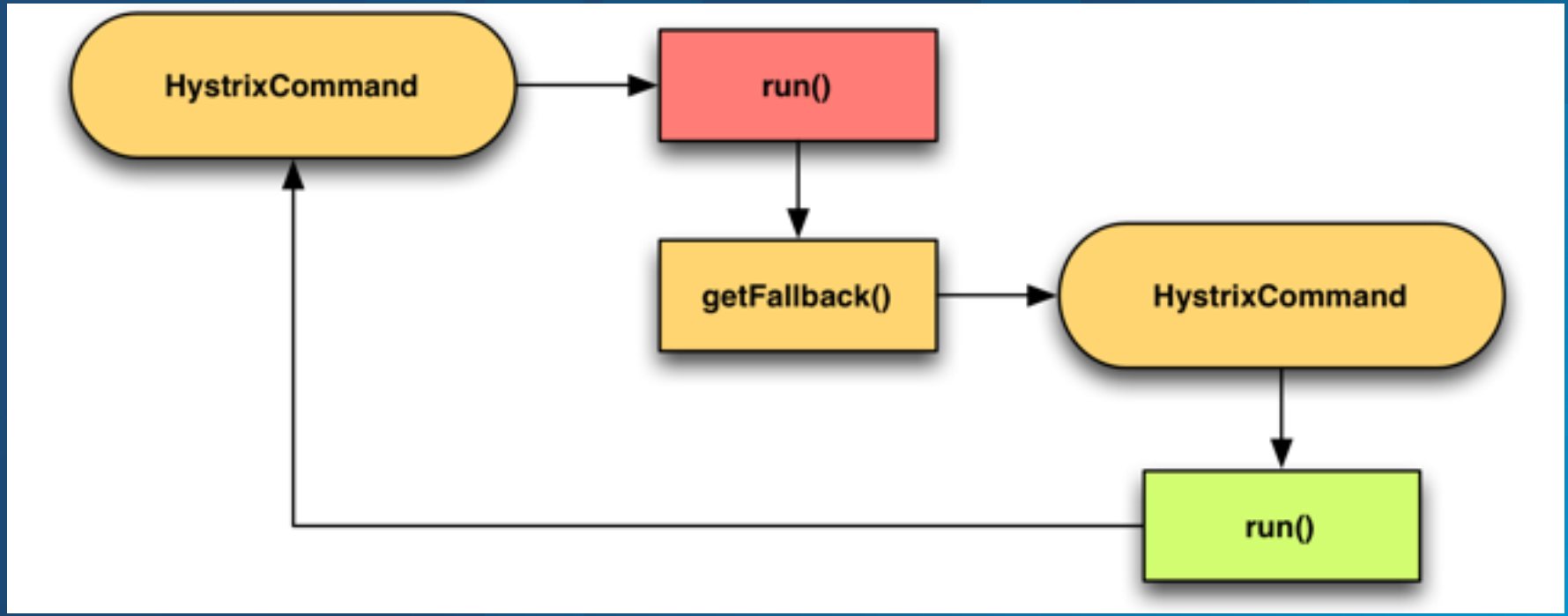
Static Fallback

```
@Override
protected Boolean getFallback() {
    return true;
}
```

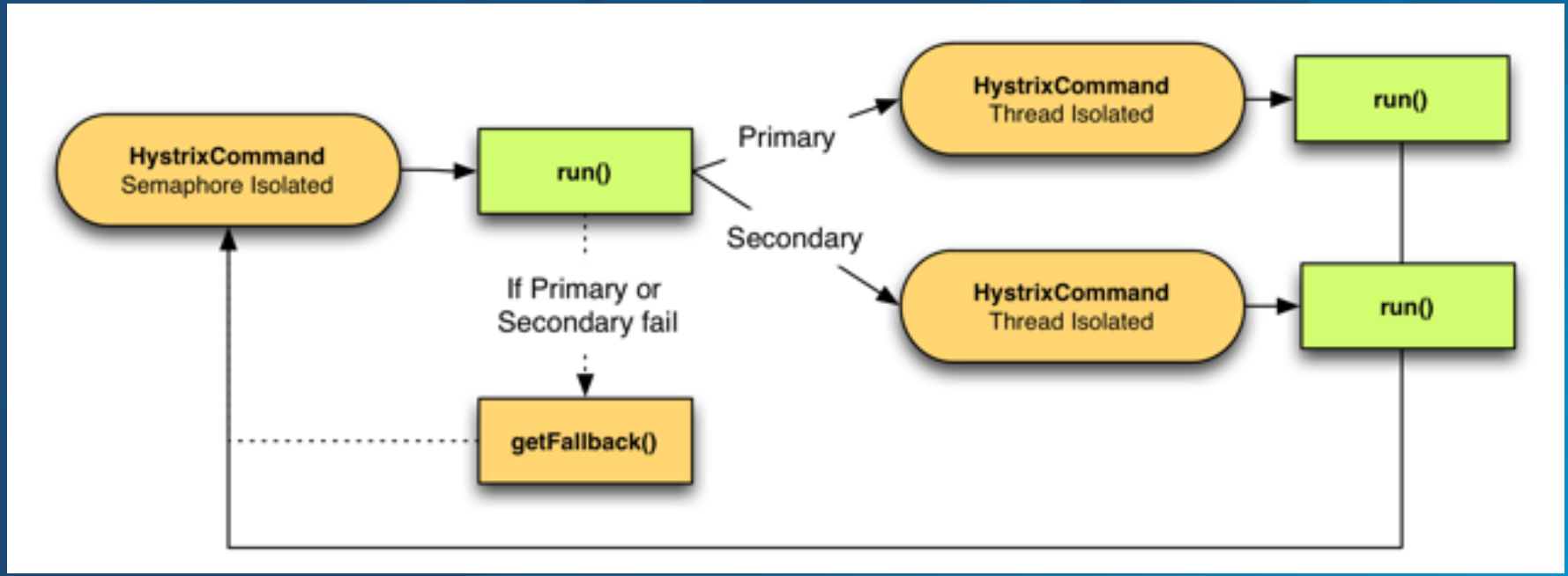
Stubbed Fallback

```
@Override
protected UserAccount run() {
    // fetch UserAccount from remote service
    //        return UserAccountClient.getAccount(customerId);
    throw new RuntimeException("forcing failure for example");
}

@Override
protected UserAccount getFallback() {
    /**
     * Return stubbed fallback with some static defaults, placeholders,
     * and an injected value 'countryCodeFromGeoLookup' that we'll use
     * instead of what we would have retrieved from the remote service.
     */
    return new UserAccount(customerId, "Unknown Name",
            countryCodeFromGeoLookup, true, true, false);
}
```

Fallback: Cache via Network

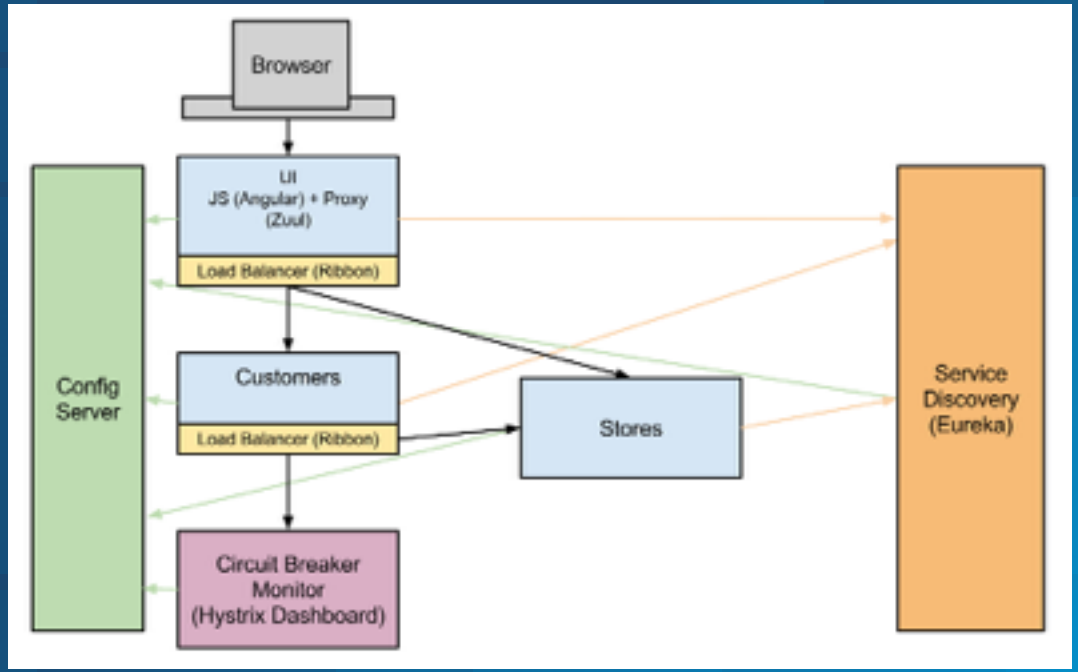


Primary + Secondary with Fallback



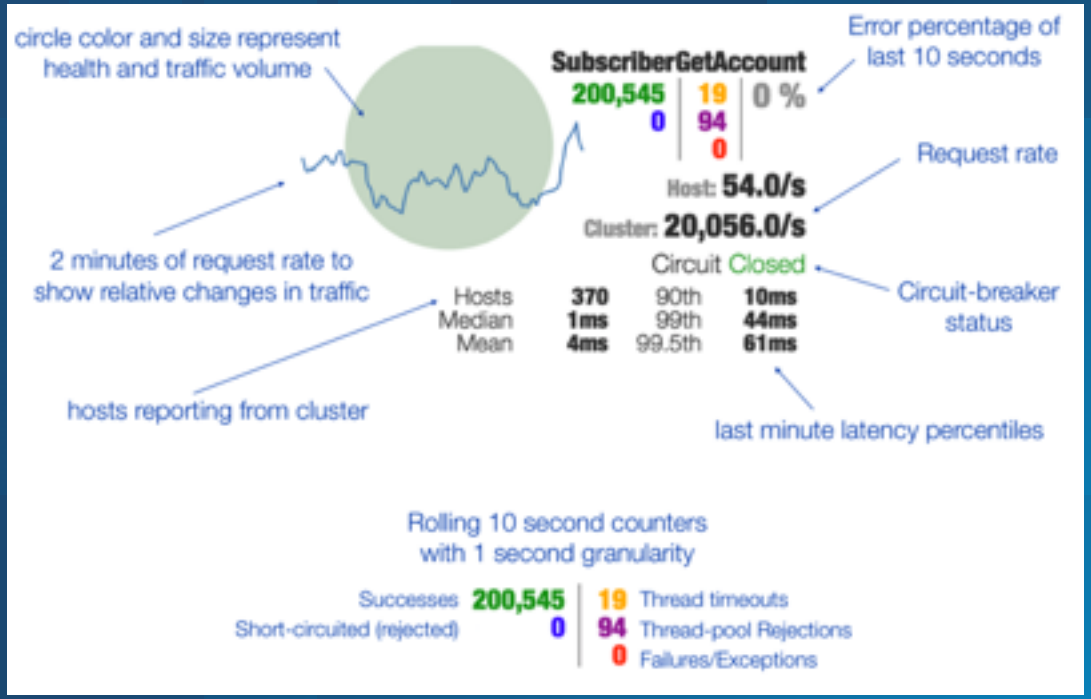
QUICK DEMO!


NETFLIX
OSS

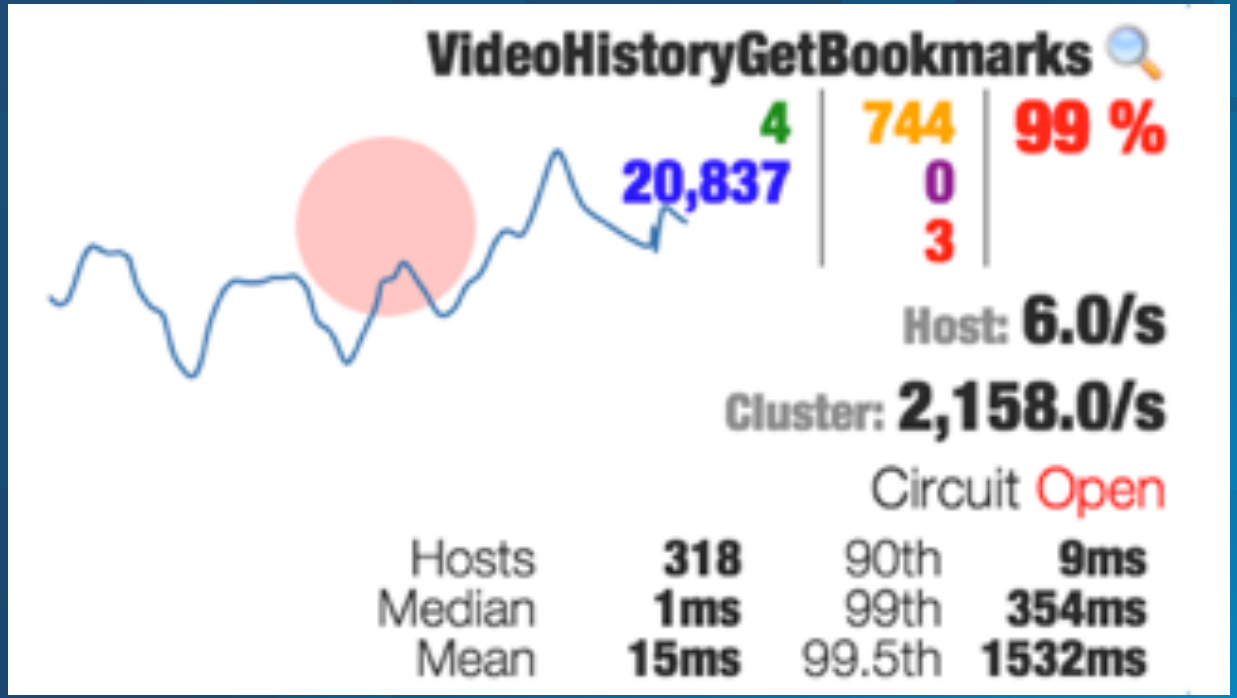

Browser
UI
JS (Angular) + Proxy
(Zuul)
Load Balancer (Ribbon)
Config
Server
Customers
Load Balancer (Ribbon)
Slores
Circuit Breaker
Monitor
(Hystrix Dashboard)
Service
Discovery
(Eureka)


CF
CF
CF
CF
CF
CF

Hystrix Dashboard



A Failing Circuit



Microservice

API Gateway

Stateless/
Shared-Nothing

Cloud Application Architecture Patterns

Thank You!!!

Matt Stine (@mstine)
Cloud Foundry Platform Engineer
matt.stine@gmail.com
http://www.mattstine.com

Configuration/
Service Consumption

Fault
Tolerance