Security In A Microservice World

Jack Mannino

Who Am I?

- Jack Mannino
  - CEO @ nVisium
  - Banned from computers by his co-workers
  - But, writes most of his (good) code in Scala
  - New York Mets and Giants fan, numb to disappointment
    - Speaking of disappointment, Go Cowboys!

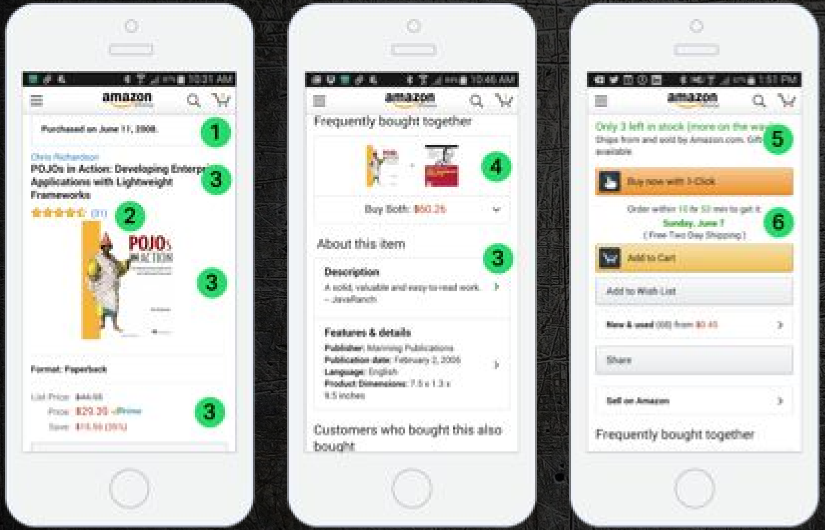
Microservices

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data. *- Martin Fowler*

SOA/ESB For Hipsters

Monolithic **_Microservices_**

One External View, Many Services



Order History

Reviews

Basic Product Info

Recommendations

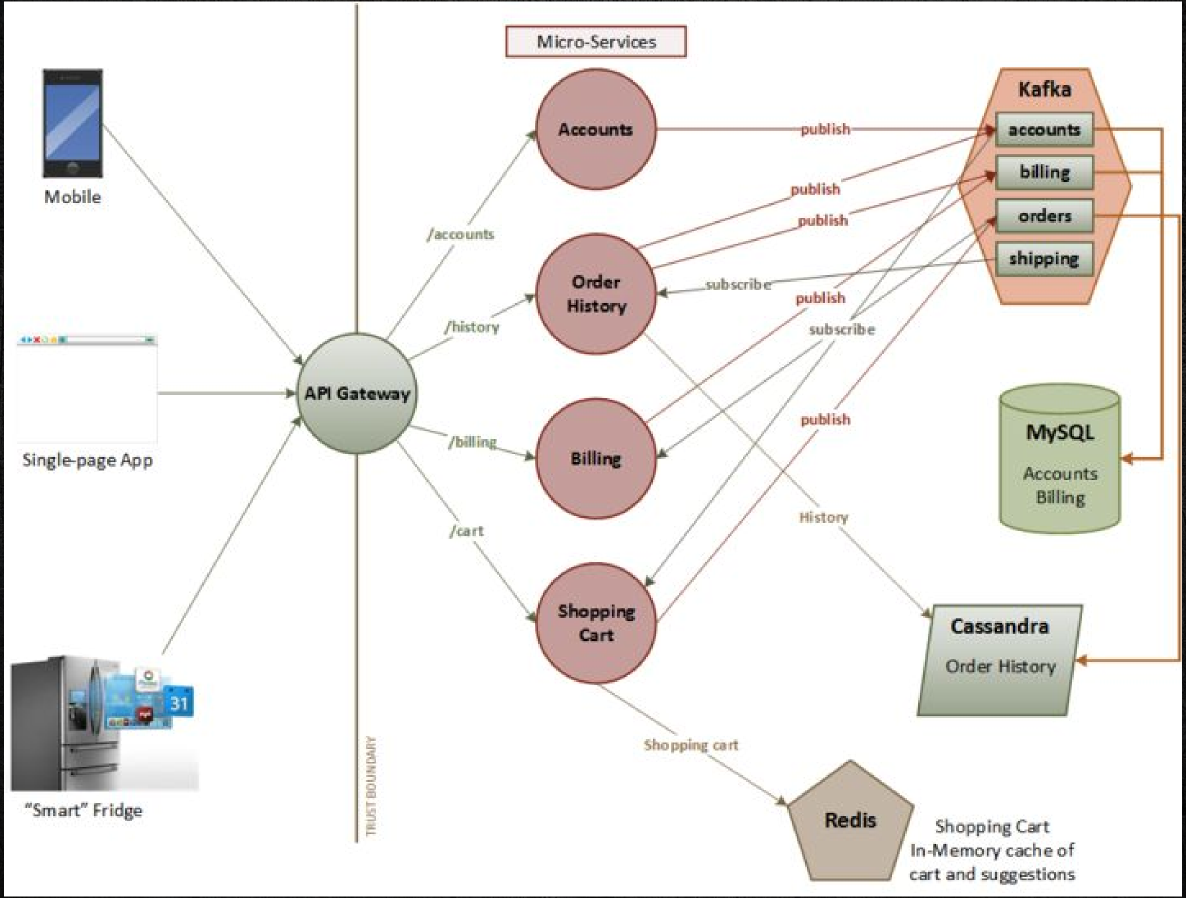
Inventory

Shipping

Microservices Pattern

- Decoupled components
- Increased complexity
- Immutable architecture
- Move faster, shorter development timeframes
  - And possibly lifetime in general
- Minimize dependencies and shared concerns
- Small and focused
- Data contracts (or not) between related services
- Less commitment to a specific technology or stack
- Good integration tests are really, really important
  - For both avoiding fragility as well as security uses

A Simple Microservice Architecture



Oh No, Security!

- Common problems amongst microservice implementations
- Sometimes these issues are developer-induced
- Sometimes there's a lack of built-in or easy security controls for a stack
- We make tradeoffs for functionality/features over security
  - Congratulations, you all have jobs.
- It's all about speed...speed, speed, speed

More Services = More Complexity

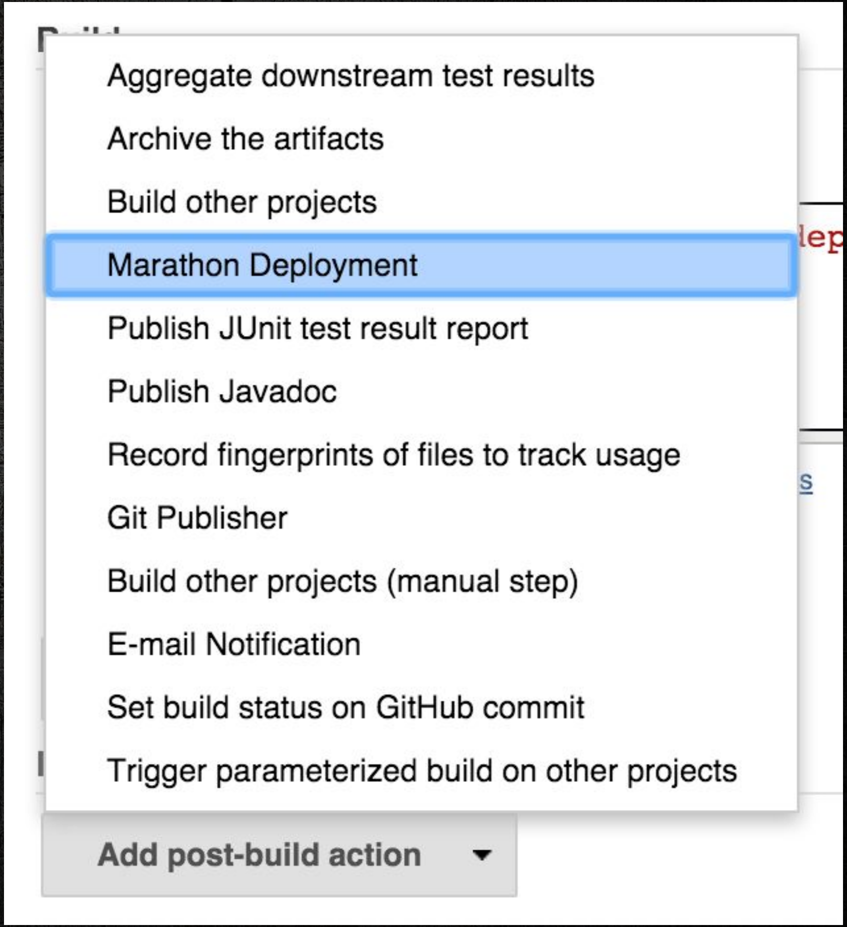
- “ A database per service”, or something like that
  - now you have more databases to manage and secure
- More complexity in deployments
  - Segmentation and isolation
  - Multi-cloud? Like some stuff in AWS, some in Azure?
  - Identity management and access control
  - Data/message integrity
  - Rapid rate of change, deprecation cycle

Your Data Is In ________?

- Where does the bouncy little ball of data go?
    - Intentional persistence
        - To a database
        - S3 bucket (hopefully not public…)
    - Unintentional persistence
        - To a log (flat or distributed)
        - Disk caching
    - Kafka
        - Time-based cleanup?
        - Based on storage quota?
            - What happens if you never hit 10 terabytes?

(Containers + Microservices) = (Peanut Butter + Jelly)

- Smaller builds
- Faster builds
- Faster deployment
- You don't want to deploy big containers
- Big containers = slower deployments
- Common pipeline workflow
    - Push to GitHub
    - Build and run tests @ Jenkins
    - Publish an updated container to Docker Hub
        - Or your container registry of choice
    - Deploy the container to <platform of choice>



Aggregate downstream test results
Archive the artifacts
Build other projects
Marathon Deployment
Publish JUnit test result report
Publish Javadoc
Record fingerprints of files to track usage
Git Publisher
Build other projects (manual step)
E-mail Notification
Set build status on GitHub commit
Trigger parameterized build on other projects
Add post-build action

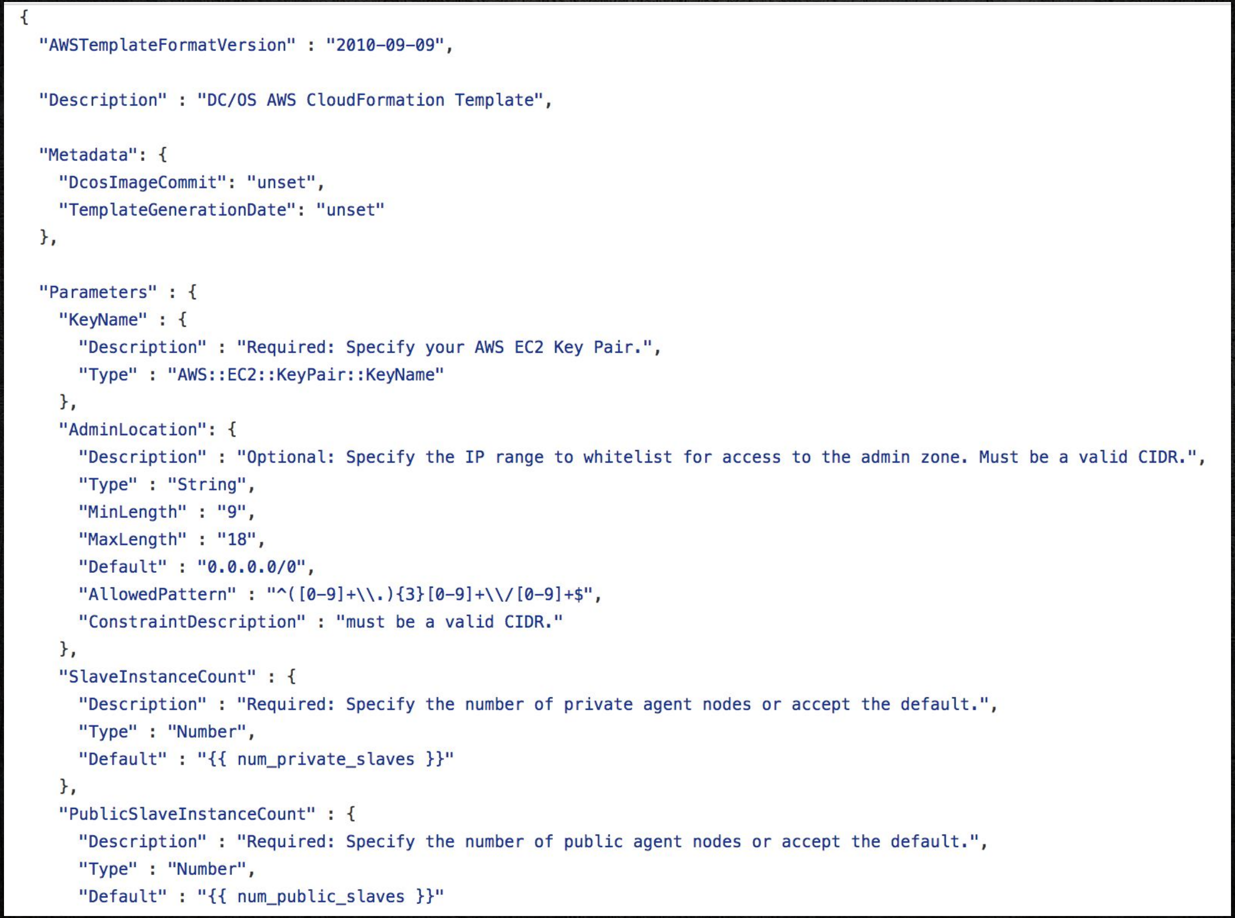
For The Cool Kids, Serverless

- AWS Lambda
- Azure Cloud Functions
- Stateless
- Often setup, processing, and teardown occurs within milliseconds
- They require you to finish within 5 minutes
- Not the best for doing batch processing
- Summarized by “finish your work in X minutes or you die”
- Security tools with a performance hit? ***Enjoy getting laughed at.***

Defining Your Architecture

- Once upon a time, your architecture looked like a bunch of servers and cables
- Now, your architecture might be in a GitHub repo
- However, one of the really nice things about infrastructure-as-code is the auditability of changes
    - Track commits (and good logging) to audit changes to your environment
- Important to restrict who can commit to master, isn't it?
- Easy to replicate, and recover an environment (assuming good backups and configuration)

CloudFormation



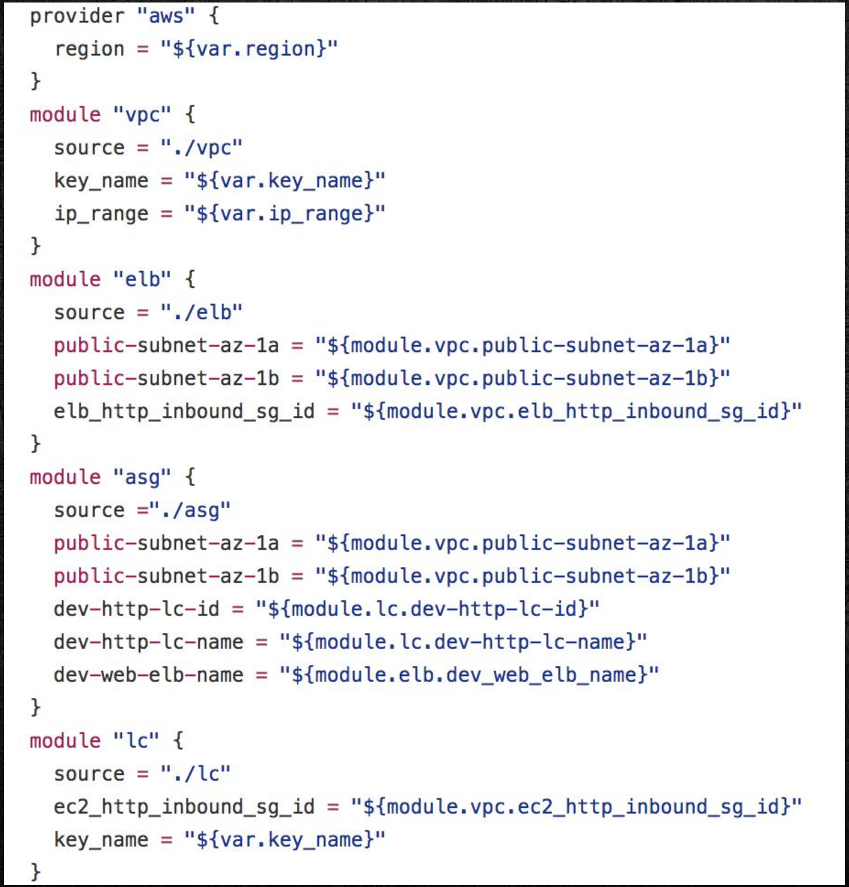
```
{
  "AWSTemplateFormatVersion" : "2010-09-09",

  "Description" : "DC/OS AWS CloudFormation Template",

  "Metadata": {
    "DcosImageCommit": "unset",
    "TemplateGenerationDate": "unset"
  },

  "Parameters" : {
    "KeyName" : {
      "Description" : "Required: Specify your AWS EC2 Key Pair.",
      "Type" : "AWS::EC2::KeyPair::KeyName"
    },
    "AdminLocation": {
      "Description" : "Optional: Specify the IP range to whitelist for access to the admin zone. Must be a valid CIDR.",
      "Type" : "String",
      "MinLength" : "9",
      "MaxLength" : "18",
      "Default" : "0.0.0.0/0",
      "AllowedPattern" : "^([0-9]+\\.){3}[0-9]+\\/[0-9]+$",
      "ConstraintDescription" : "must be a valid CIDR."
    },
    "SlaveInstanceCount" : {
      "Description" : "Required: Specify the number of private agent nodes or accept the default.",
      "Type" : "Number",
      "Default" : "{{ num_private_slaves }}"
    },
    "PublicSlaveInstanceCount" : {
      "Description" : "Required: Specify the number of public agent nodes or accept the default.",
      "Type" : "Number",
      "Default" : "{{ num_public_slaves }}"
```

Terraform



```
provider "aws" {
  region = "${var.region}"
}
module "vpc" {
  source = "./vpc"
  key_name = "${var.key_name}"
  ip_range = "${var.ip_range}"
}
module "elb" {
  source = "./elb"
  public-subnet-az-1a = "${module.vpc.public-subnet-az-1a}"
  public-subnet-az-1b = "${module.vpc.public-subnet-az-1b}"
  elb_http_inbound_sg_id = "${module.vpc.elb_http_inbound_sg_id}"
}
module "asg" {
  source ="./asg"
  public-subnet-az-1a = "${module.vpc.public-subnet-az-1a}"
  public-subnet-az-1b = "${module.vpc.public-subnet-az-1b}"
  dev-http-lc-id = "${module.lc.dev-http-lc-id}"
  dev-http-lc-name = "${module.lc.dev-http-lc-name}"
  dev-web-elb-name = "${module.elb.dev_web_elb_name}"
}
module "lc" {
  source = "./lc"
  ec2_http_inbound_sg_id = "${module.vpc.ec2_http_inbound_sg_id}"
  key_name = "${var.key_name}"
}
```

Segmentation

- Segmentation looks different when it's configuring a stack of switches, pressing a few buttons in the AWS console, or if we're programmatically defining these things
- Your architecture and goals define what's acceptable
  - E.g.- Bounded Context, where multiple related services have some common components
  - Limiting the services that can publish or subscribe for various data feeds
- Common sense dictates the rest
  - Do all of your microservices need write capabilities to Redis?
  - Can they suffice with a read-only slave or replica instead?
  - How easy is it for a simple misconfiguration or architecture change to burn your universe down?

But Isn’t This An Enterprise Service Bus? No.

- In comparison, traditional ESBs have tons of dependencies
  - Enterprise message model requires strict, rigid structure
  - Updating the model requires updating multiple services
- Autonomous services, with their own lifecycle
  - More suited to anti-monolithic design
- Routing and service discovery works a bit different
  - Much more dynamic, tools like Consul are useful https://www.consul.io/
  - Framework, infrastructure, and platform woes
    - Like Lagom + Kubernetes, for instance
      - Challenges with discovering services using Kubernetes https://github.com/lagom/lagom/issues/59
      - Requires https://github.com/typesafehub/service-locator-dns
    - AWS ECS provides a little bit, and you have to hook up the rest
      - Recently, more service discovery features
      - Lagging behind Kubernetes, DC/OS, Docker Swarm, etc.

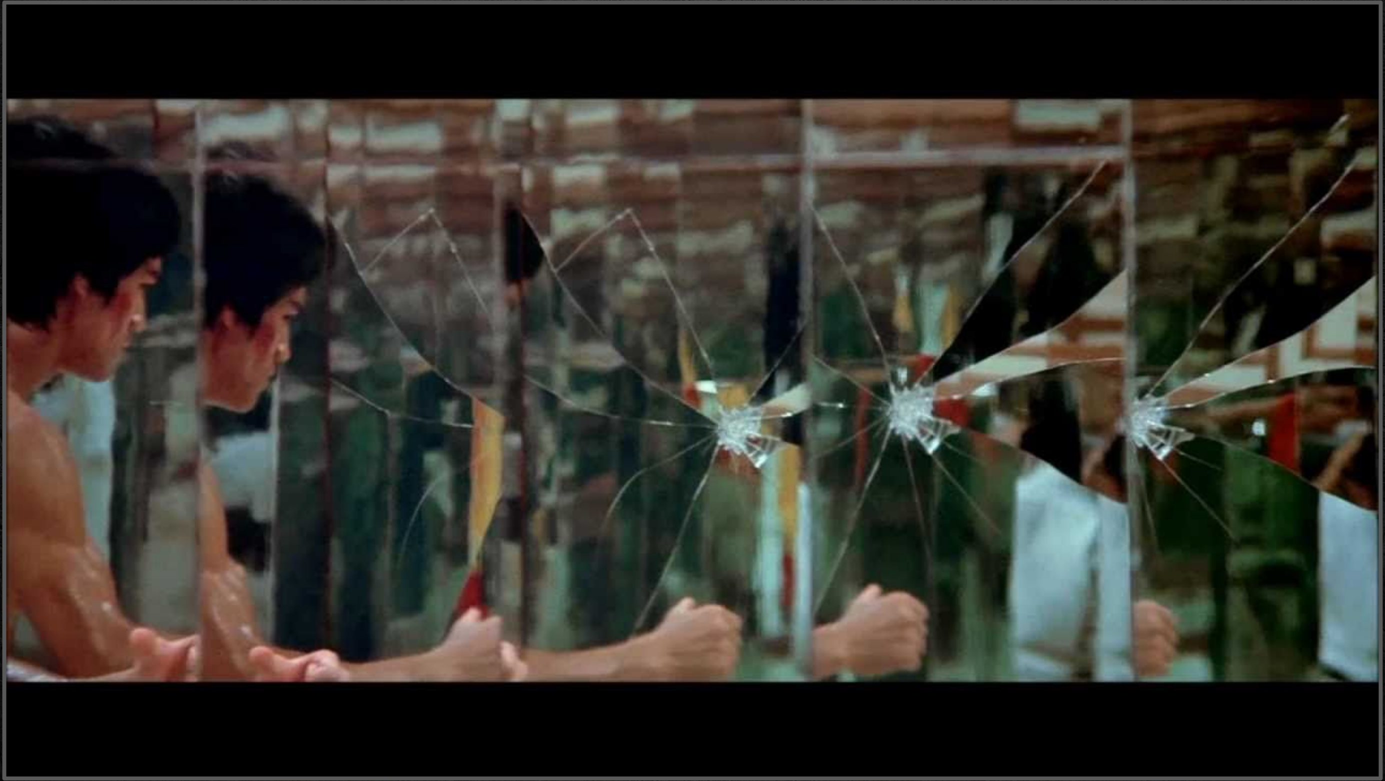
Step #1 - Secure Your APIs and Apps

- Your APIs are the gateway into the microservice architecture
- First step, general API security hygiene
- Nothing new here...OWASP Top 10, SomeList Top 100, whatever
- SQL Injection is still the same, XSS is still XSS if you do rendering, etc.
  - Trust no external data...validate/encode/sanitize accordingly...all of that
- Anyone selling you Cross Service Microservice Injection is full of shit
- Authentication and authorization require new considerations

Safe Data Handling + Serialization/Deserialization

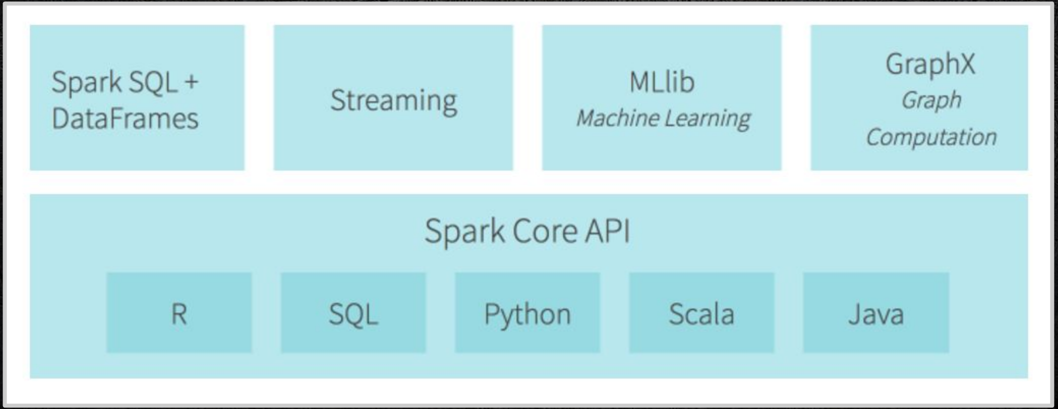
- Another service or consumer may have different intentions than you did
- Is your data safe to use against *any* interpreter
  - Probably not
- Do you understand how every single consumer will leverage your data?
  - Probably not
- Attackers can exploit this
  - Examples
    - Spark SQL
    - Kafka + Spark Serialization

Spark SQL



Example - Spark Streaming + Spark SQL

- Spark helps demonstrate some of the challenges with traditional testing
- Open source general purpose cluster computing framework
  - Spark Core, Spark Streaming, Spark SQL, GraphX, and MLLib
- Commercial support and a large ecosystem
- Better suited for stream processing and iterative machine learning algorithms than Hadoop/MapReduce
- Why do we care? **We develop software with Spark.**

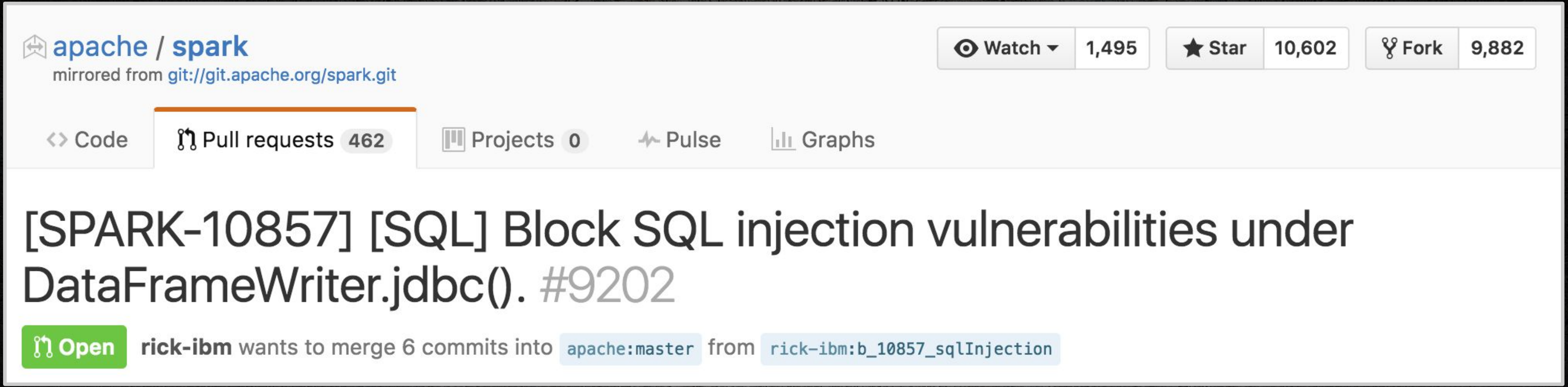


nvisium

Spark SQL

- Your data becomes SQL, somehow
- Spark SQL allows you to query structured data from many sources
    - JDBC (MySQL, PostgreSQL)
    - Hive
    - JSON
    - Parquet
    - Optimized Row Columnar (ORC)
    - Cassandra
    - HDFS
    - S3
- Catalyst query compiler introduced in Spark 2.0
    - ANTLR Grammar for SQL, including Spark-specific items
    - Abstract Syntax Tree builder converts ANTLR ParseTrees into expressions
- Lexical analysis for syntax verification and query construction

Spark SQL—It's You, Not Me

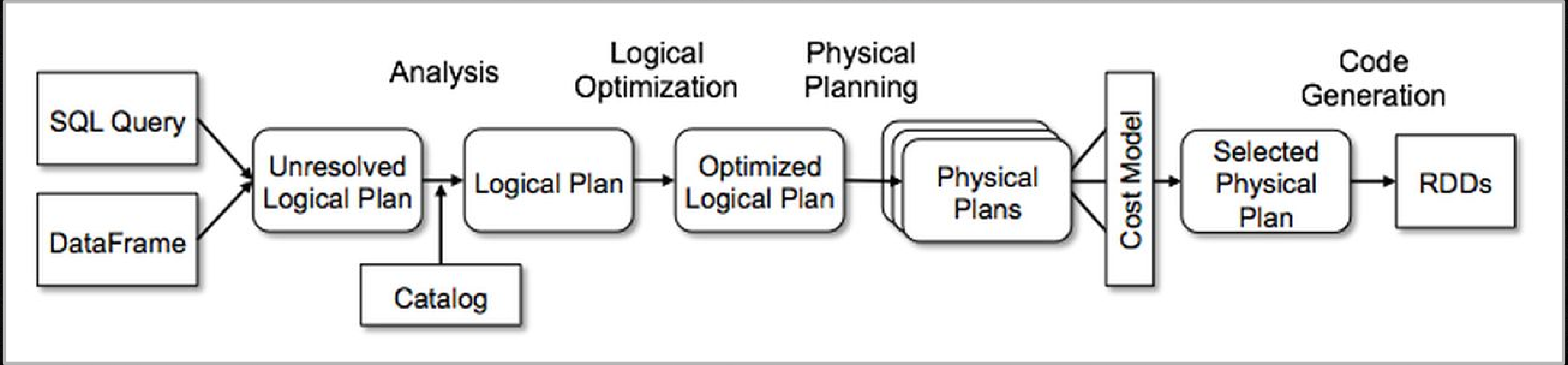

apache / spark
mirrored from git://git.apache.org/spark.git

[SPARK-10857] [SQL] Block SQL injection vulnerabilities under DataFrameWriter.jdbc(). #9202

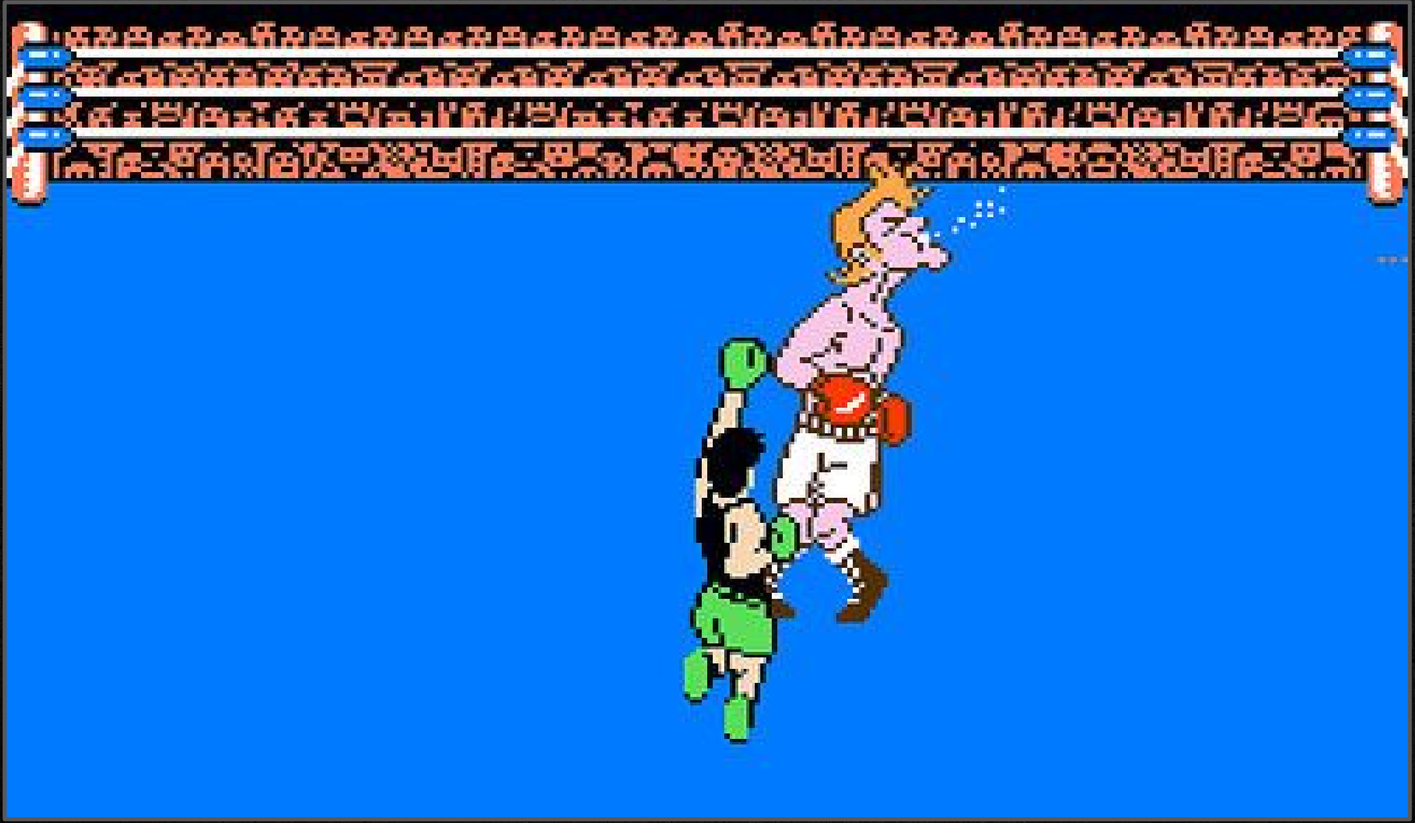
Open rick-ibm wants to merge 6 commits into `apache:master` from `rick-ibm:b_10857_sqlInjection`

"If it's simple, quoting sounds good. This isn't really a SQL injection problem since it would be up to callers to sanitize inputs from an external source, and Spark is not something you would expose directly to external calls or input. Still this may avoid corner case problems like table names with a special character."

Spark SQL + Catalyst



Serialization and the Questionable Chin



Spark <3s Serialization

- Serialization and deserialization happens everywhere
- Speed is paramount
- Lots of inference to deduce class signatures on-the-fly

Unsafe Deserialization

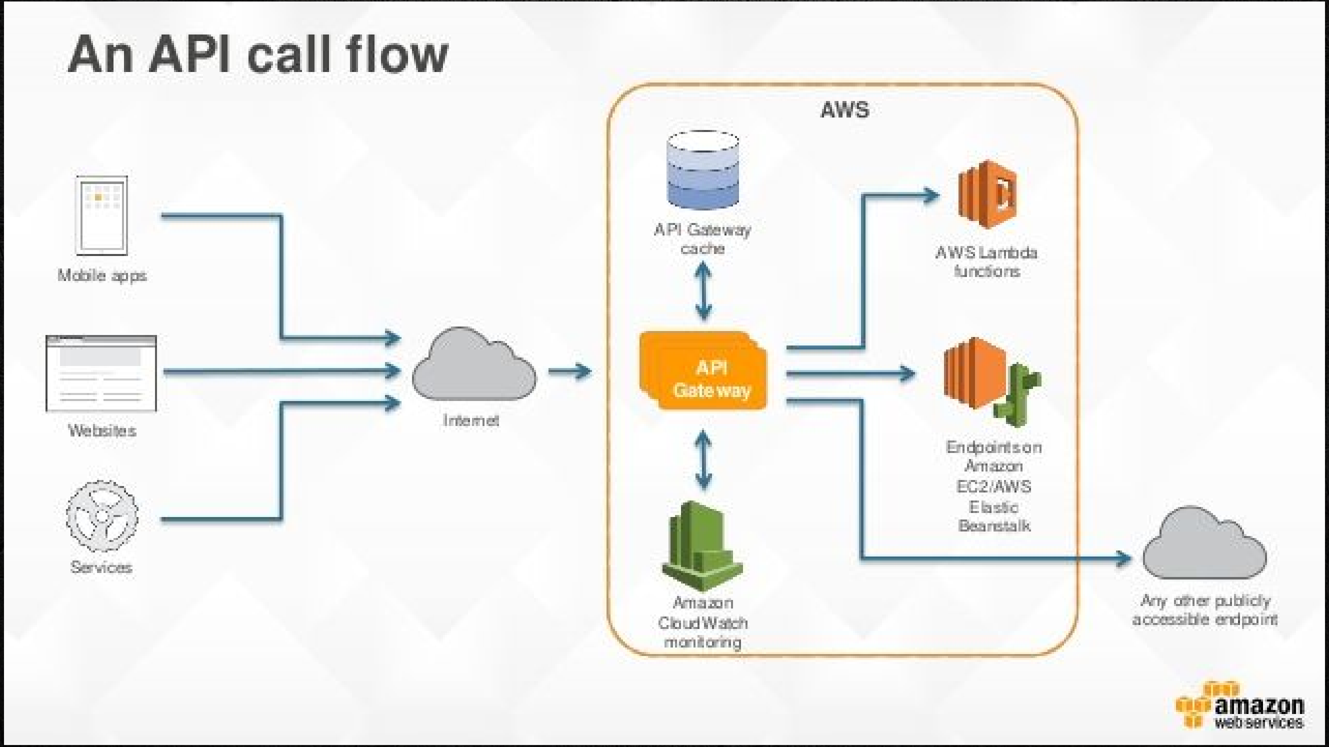
- Kryo is the default serializer
    - Java serialization is also used, too
- By default, Kryo disables type registration
    - Remember, speed is paramount right?
    - *private val registrationRequired = conf.getBoolean("spark.kryo.registrationRequired", false)*
    - *kryo.setRegistrationRequired(registrationRequired)*
- *spark.kryo.registrationRequired*
- *spark.kyro.classesToRegister*
- Open issue: https://github.com/EsotericSoftware/kryo/issues/398

Unsafe Deserialization

- What happens when you lie to Kryo (or other serializers)?
- Invoke anything loaded on the classpath
  - Call zero-argument constructors
  - Call finalize functions
- Example of a few tricks against Kryo
  - https://www.contrastsecurity.com/security-influencers/serialization-must-die-act-1-kryo
  - https://github.com/Contrast-Security-OSS/serialbox/blob/master/src/test/java/com/contrastsecurity/serialbox/KryoTest.java
- Spark consumes serialized data within developer APIs and internal APIs

Authentication

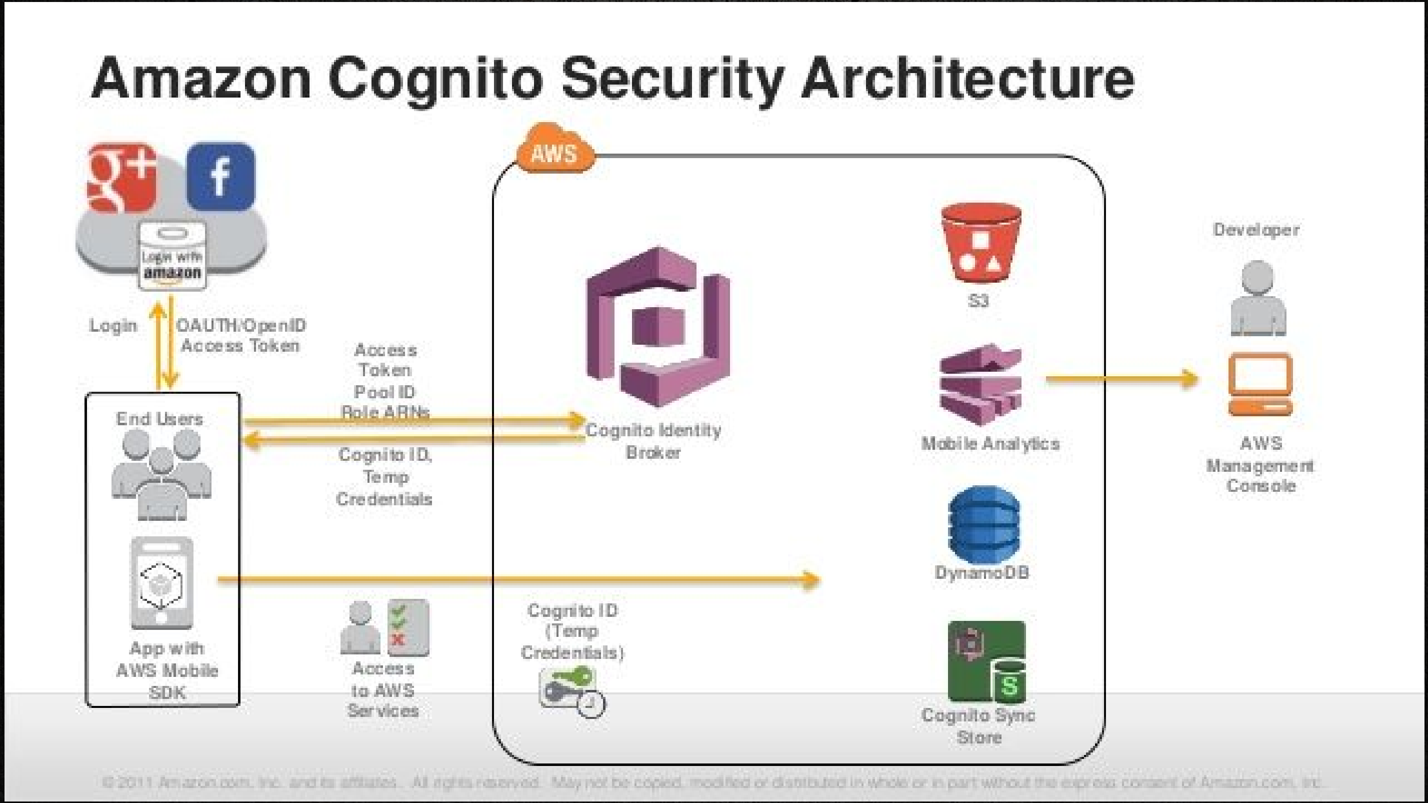
- One popular pattern is the API Gateway Pattern
- Handles authentication and provides service discovery



Authentication

- At my day job, we use Cognito + AWS API Gateway to handle the heavy lifting
    - https://aws.amazon.com/blogs/mobile/integrating-amazon-cognito-user-pools-with-api-gateway/
    - Cognito handles authenticating with credentials, MFA, etc.
- API Gateway checks your access token or JWT (meh) and grants access
- Perform role-based restrictions across services
- Each request is signed, which provides an additional layer of authentication
    - https://docs.aws.amazon.com/apigateway/api-reference/signing-requests/
    - Integrate Lambda functions for pre/post processing hooks
- Can consume your Swagger files
    - Your framework probably makes it easy to produce Swagger files
    - We use https://github.com/iheartradio/play-swagger

Authentication

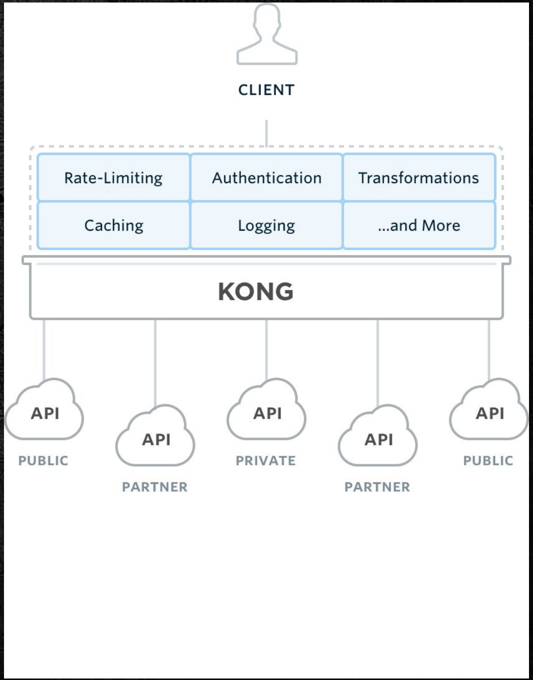

Amazon Cognito Security Architecture
Login with amazon
Login
OAUTH/OpenID Access Token
End Users
App with AWS Mobile SDK
Access Token Pool ID Role ARNs
Cognito ID, Temp Credentials
Access to AWS Services
AWS
Cognito Identity Broker
Cognito ID (Temp Credentials)
S3
Mobile Analytics
DynamoDB
Cognito Sync Store
Developer
AWS Management Console
© 2011 Amazon.com, Inc. and its affiliates. All rights reserved. May not be copied, modified or distributed in whole or in part without the express consent of Amazon.com, Inc.

Once You Get Past The API Gateway . . .

- The gateway can share data with downstream services
- Authorization strategies
  - Post-authentication Lambda function to check role/entitlements
- Custom attributes
  - Default stuff like username, email
  - Custom attributes to suit your needs
- Replicating the data (cache, other storage)

Authentication

- Kong allows you to centralize authentication and abstract it away from your services, using credentials, JWT, API tokens etc.
- Authorization provider for OAuth and others
- Explicitly grant users (Consumers) access to services
- https://github.com/Mashape/kong
- Simple to setup and quick configuration
- Things that suck:
    - Plugins that use SHA1 for password hashing :-(
    - Maintaining your own plugins because the defaults suck
- If I have to maintain my own secure plugin for something that they should be doing, it's not for me



Quick, Dirty Authentication

- Example of implementing authentication for multiple services using Kong

```
curl -i -X POST   --url https://localhost:8001/apis/   --data 'name=nv-api'   --data "upstream_url=https://$1:10000/api"
--data 'request_path=/api'

curl -i -X POST   --url https://localhost:8001/apis/   --data 'name=nv-login'   --data
"upstream_url=https://$1:11000/login"   --data 'request_path=/login'

curl -i -X POST   --url https://localhost:8001/apis/   --data 'name=nv-register'   --data
"upstream_url=https://$1:12000/register"   --data 'request_path=/register'

curl -X POST https://localhost:8001/apis/nv-login/plugins --data "name=bcrypt-auth"

curl -X POST https://localhost:8001/apis/nv-api/plugins --data "name=key-auth"
```

Access Control and Authorization

- If things are decentralized, how the hell do we do that?
- Depending on the size of your development shop, autonomous teams that probably don’t talk to each other
- Roles and entitlements mean different things across services
- Users have access to different things
- Things have relationships with other things
- Often, many more potential attack paths than in a monolithic architecture

Access Control and Authorization

- Options
  - Restrict at the API Gateway, pass user principal data to services
  - Delegation with OAuth
  - SAML (for the masochists in the audience)
  - Various libraries and authorization frameworks
- Roll your own
  - It might actually make sense? Maybe?

Access Control and Authorization

- **Strategies**
  - ***Attempt to centralize it somehow***
    - The question is what can you centralize?
    - Object-level entitlements?
    - API endpoints?
    - At the method level?
    - Message level authentication and signing?
  - ***Centralize what you can***
    - From experience, common roles can be enforced if they exist globally
      - Org admin
      - Billing administrator
    - Service-level granularities are best left to the layer that understands them
  - ***Don’t centralize anything***
    - Good luck
    - YOLO?

Access Control and Authorization

- Centralized
- Focus on what you can see/understand right behind API Gateway
- You know who the user is, what role they have, and all of that
- You might not know which specific objects they have access to in a database, though (maybe you do)
- You don’t know the nitty gritty business rules of a given service (unless you’ve already extracted them to a Lambda function)

Access Control and Authorization

- Centralized where it makes sense
- Allow a globally consistent role to be used and as the first line of defense against unauthorized access to a given microservice
- Example - without the “admin” role, you can’t hit */admin* endpoints
- Let each service enforce the appropriate rules, as it knows best
- Obvious dangers: your developers don’t talk to each other
- Obvious dangers: you don’t have a sane way to enforce consistency

What About Authorization BETWEEN Services?

- Think of scenarios such as using a shared message queue between services
  - Who can publish to topics?
  - Who can publish to specific partitions (ie- Kafka)
  - Who else can consume your data?
- Who administers the topics? Who approves access?

Producer:

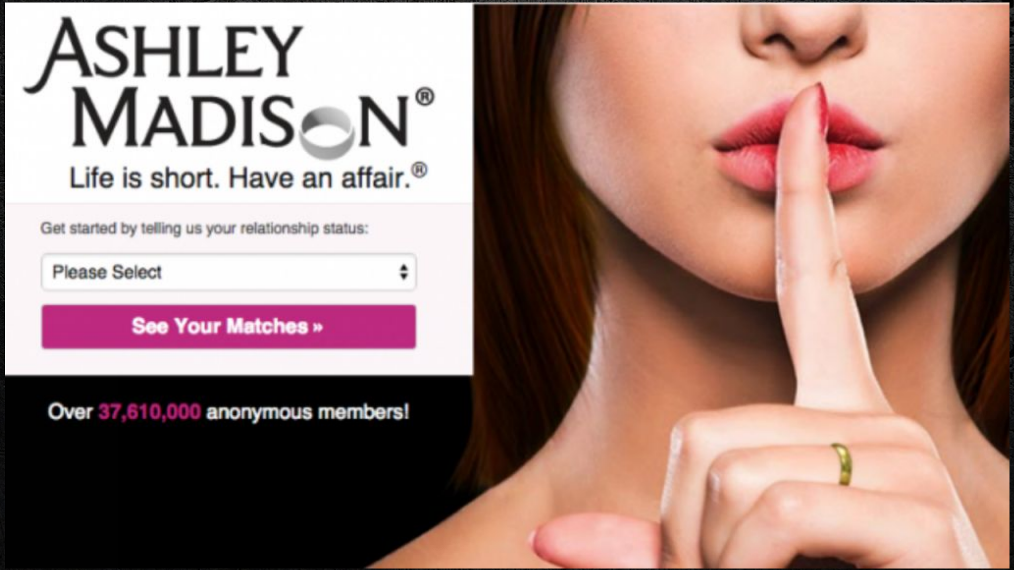
```
kafka-console-producer.sh --broker-list localhost:9092 --topic creditcard-stuff
This is a credit card # 1234567890123456
This is a credit card # 1234567890111111
```

Consumer:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic
creditcard-stuff --from-beginning
```

Securely Sharing Secrets

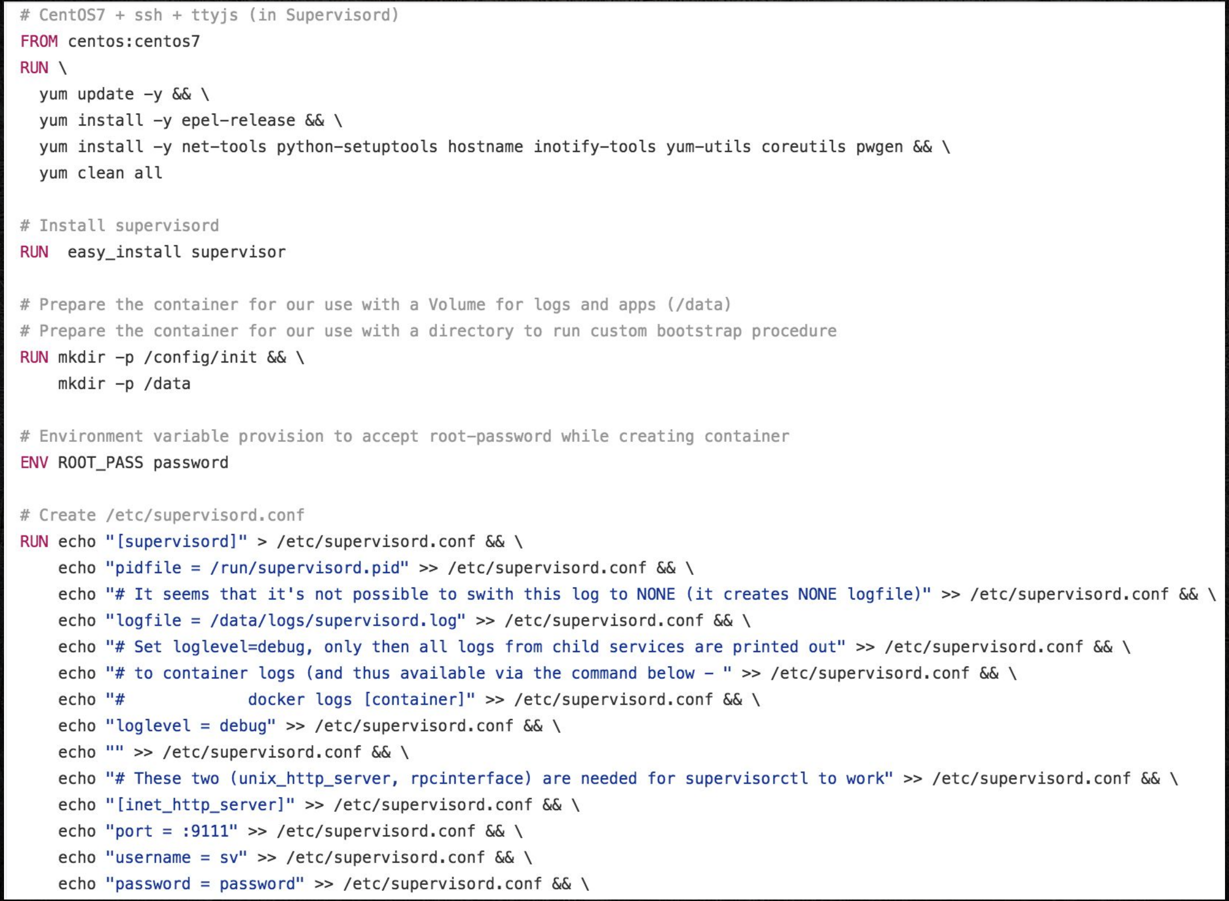
- There are options, but you have to actually use them
- You don't want to pass credentials around in the clear
- Some container orchestration platforms are better than others


ASHLEY MADISON®
Life is short. Have an affair.®
Get started by telling us your relationship status:
Please Select
See Your Matches »
Over 37,610,000 anonymous members!

Bad Ideas

- Hardcoding credentials into your source code
- Hardcoding credentials into your Dockerfile
- Using environment variables as a crutch for passing secrets
    - Environment variables are available to each process in the container

Harcoded Secrets Via Dockerfile :-(



```
# CentOS7 + ssh + ttyjs (in Supervisord)
FROM centos:centos7
RUN \
  yum update -y && \
  yum install -y epel-release && \
  yum install -y net-tools python-setuptools hostname inotify-tools yum-utils coreutils pwgen && \
  yum clean all

# Install supervisord
RUN  easy_install supervisor

# Prepare the container for our use with a Volume for logs and apps (/data)
# Prepare the container for our use with a directory to run custom bootstrap procedure
RUN mkdir -p /config/init && \
    mkdir -p /data

# Environment variable provision to accept root-password while creating container
ENV ROOT_PASS password

# Create /etc/supervisord.conf
RUN echo "[supervisord]" > /etc/supervisord.conf && \
    echo "pidfile = /run/supervisord.pid" >> /etc/supervisord.conf && \
    echo "# It seems that it's not possible to swith this log to NONE (it creates NONE logfile)" >> /etc/supervisord.conf && \
    echo "logfile = /data/logs/supervisord.log" >> /etc/supervisord.conf && \
    echo "# Set loglevel=debug, only then all logs from child services are printed out" >> /etc/supervisord.conf && \
    echo "# to container logs (and thus available via the command below - " >> /etc/supervisord.conf && \
    echo "#            docker logs [container]" >> /etc/supervisord.conf && \
    echo "loglevel = debug" >> /etc/supervisord.conf && \
    echo "" >> /etc/supervisord.conf && \
    echo "# These two (unix_http_server, rpcinterface) are needed for supervisorctl to work" >> /etc/supervisord.conf && \
    echo "[inet_http_server]" >> /etc/supervisord.conf && \
    echo "port = :9111" >> /etc/supervisord.conf && \
    echo "username = sv" >> /etc/supervisord.conf && \
    echo "password = password" >> /etc/supervisord.conf && \
```

Secrets Via Environment Variables

```
docker run –it –e “DBUSER=dbuser” –e “DBPASSWD=dbpasswd” mydbimage
```

When you do this, every process in your container can read these variables

You also risk leaking passwords via logs and various dashboards

Securely Sharing Secrets

- Kubernetes -> native secrets management API
  - http://kubernetes.io/docs/user-guide/secrets/
- Docker/Swarm -> recent support
  - Secret Management https://github.com/docker/docker/pull/27794
  - https://www.vaultproject.io/
- DC/OS - Secrets API available in 1.8+ Enterprise
  - https://docs.mesosphere.com/1.8/administration/secrets/secrets-api/

Passing Secrets To Kubernetes Pods

Create Files

```
$ echo -n "administrator" > ./username.txt
$ echo -n "0XDEADB33F" > ./password.txt
```

Create Secrets

```
$ kubectl create secret generic db-user-pw --from-file=./username.txt
--from-file=./password.txt secret "db-user-pw" created
```

Use a Secret

```
"apiVersion": "v1",
 "kind": "Pod",
  "metadata": {
   "name": "jackpod",
   "namespace": "jack"
  },

  "spec": {
   "containers": [{
    "name": "cart-cache",
    "image": "redis"
    "volumeMounts": [{
     "name": "redis-secrets",
     "mountPath": "/etc/redis-secrets",
     "readOnly": true
    }]

   "volumes": [{
    "name": "foo",
    "secret": {
     "secretName": "mysecret"
    }
   }]
  }
}
```

But Even The Good Implementations . . .

- By default, Kubernetes stores your credentials in plain text on the server in **etcd**
  - **etcd** is a distributed key-value store
  - It stores and replicates cluster state
- However, it securely passes them into the container at runtime
- But, it’s a mounted volume within your container
- Solves securely passing secrets, but still weakly protected at rest and runtime

Securely Sharing Secrets - Vault

- Store data
  - *vault write -address=${VAULT_ADDR} \
    secret/api-key value=12345678*
- Read data
  - *vault read -address=${VAULT_ADDR} \
    secret/api-key*
- Read data via API
  - *curl -H "X-Vault-Token:$VAULT_TOKEN" \
    -XGET https://docker:8200/v1/secret/api-key*

Rotate Your Keys, Maybe?

- While we’re on the topic of securely sharing secrets…
- Do you ever actually rotate your secrets?
- On a schedule? Once-in-a-while? Never?
- You can simplify some of these rituals using Vault and AWS Key Management Services (KMS)

Summary

- No-brainer that you’re better off thinking about these things as early as possible
- It’s very easy to overcomplicate your security architecture and design
- Group security tasks and controls where it makes sense
- Keep track of who has access to which systems
- Keep track of which systems can communicate with other systems
- Keep track of your data and how it’s persisted, backed up, retained etc.

Keep In Touch!

Jack Mannino
@jack_mannino
jack@nvisium.com