Wiredcraft

Graduating your node.js API to production environment

Learn about

Scalable code / Architecture / Servers

Goal for this talk

Give you confidence to take your API's to production.

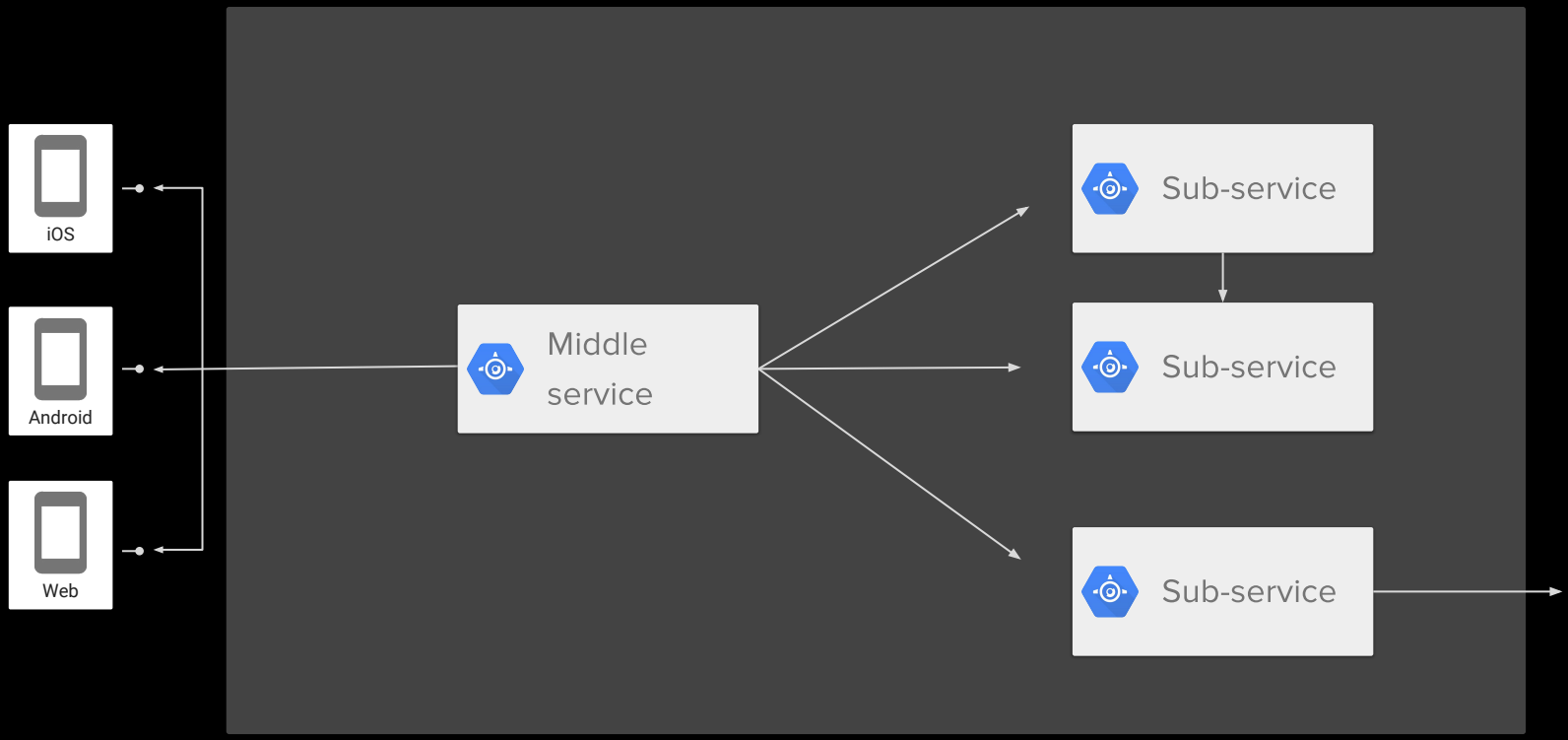
Goal for this talk

Give you confidence to take your API's to production.

...or

scare you from ever launching

Type of architecture we expect



Adjust your thinking

What does production system mean

System that has real users and real data. Typically at least few thousand daily users for public services.

Thinking

What does ‘being production ready’ mean

Thinking

What does ‘being production ready’ mean

Developer: “Code runs and functionality passes tests

Thinking

What does ‘being production ready’ mean

Developer: “Code runs and functionality passes tests

Business Manager: “System works and adds value to users and brings profit”

Thinking

What does ‘being production ready’ mean

Developer: “Code runs and functionality passes tests

Business Manager: “System works and adds value to users and brings profit”

Library developer: “Project is widely adopted and well documented”

Thinking

What does ‘being production ready’ mean

Developer: “Code runs and functionality passes tests

Business Manager: “System works and adds value to users and brings profit”

Library developer: “Project is widely adopted and well documented”

Devops guy: “Runtime environment is stable, debuggable and maintainable”

Thinking

What does ‘being production ready’ mean

Developer: “Code runs and functionality passes tests

Business Manager: “System works and adds value to users and brings profit”

Library developer: “Project is widely adopted and well documented”

Devops guy: “Runtime environment is stable, debuggable and maintainable”

Security Expert: “System is checked for common security threats and critical data is encrypted.”

Avoid missing responsibilities

Make sure that your team as a whole covers all the responsibilities needed for going to production including: Components, Code Quality, Performance, Security, Release management, User Experience

Example of a common reason for failure

"Logging component is responsibility of Joe, who left the project, it should be working, but we don't really know how it works."

Learnings:

- Make sure all parts of the codebase are looked after
- Make sure the transfer of responsibility is done properly

Code

Requirements for making your code production ready

- Stable
- Efficient
- Debuggable

Code

Logging

Devops guy's definition of production ready: "Runtime environment is stable, **debuggable** and maintainable"

Logging

Devops guy's definition of production ready: "Runtime environment is stable, **debuggable** and maintainable"

General philosophy:

Logs should tell a story.

You need a logging platform

You need a central service that can collect and expose the logs with powerful search, filtering and visualization capabilities.

ELK (Elasticsearch, Logstash, Kibana) is common stack.

There are many cloud services such as Loggly, Splunk, New Relic

Code

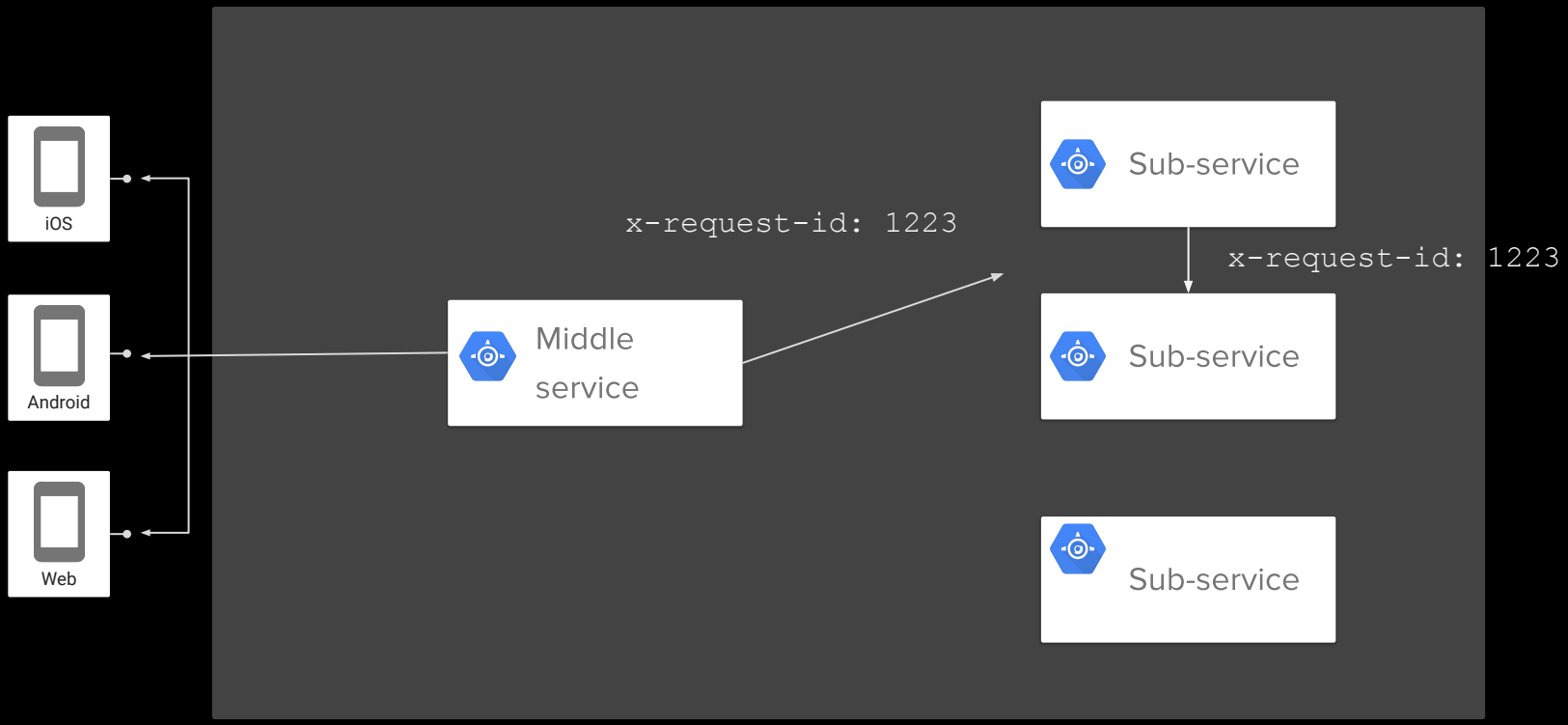
How to track logs across components?

Introduce concept of request-id's:

- When request first enters the system (at the gateway for example) it generates uuid and attaches it as a `x-request-id` http header
- Upstream services will read that value and for every log line they attach the request-id
- If they make any requests to second level services they attach the `x-request-id` header.
- Devops guys stay happy since they can now follow the flow of requests through the distributed system.

Code

How to track logs across components



Using debug is not enough!

Debug library is a powerful and easy to use library, it however is not designed for production systems since it doesn’t offer multiple log levels!

We recommend using an alternative logger (Winston for example) for application logs. This should:

- Include logging incoming requests and their information
- Any operations that modify data
- Requests and responses to/from external services

Code

Using debug is not enough!

You can still use debug (any you probably should!) on the same codebase for development level logging.

Expect external services to fail

How will your API behave if upstream API is:

- Taking too long to respond?
- Rejecting connections?
- Giving application errors?

Expect external services to fail

Will your API:

- Get unresponsive? (bad)
- Recover automatically when API recovers? (a must)
- Leave data in inconsistent state? (very bad)
- Start returning 5xx errors and stay responsive? (sometimes the best option)

How to survive upstream service failures

- Design the business logic accordingly
- Have proper error handlers for different cases
- Manage your timeout limits consciously.
- Limit maximum amount of connections to a service
- If needed, Circuit Breaker pattern to the rescue...

Code

Circuit Breaker pattern

Try to detect if service is down and stop calling it if it is down

- Have “counter” to keep track of failures for a service
- If limit is exceeded stop calling the service and just return an error immediately

Helps to stop bombing upstream service that is trying to recover and respond faster when they are

Code

Circuit Breaker example (brakes package)

```
const brake = new Brakes(serviceCall, {
  statInterval: 2500,
  threshold: 0.5,
  circuitDuration: 15000,
  timeout: 250
});

brake.fallback(() => {
  return Promise.resolve('Service not available');
});

brake.exec().then(...)
```

Hystrix

If you are interested to learn more you should check Hystrix by Netflix:

https://github.com/Netflix/Hystrix/wiki

The project is in Java, but the wiki if useful learning resource.

Brakes (and many others) are based on concepts from Hystrix.

Code

Common pattern in node.js is to do something like:

```
function fetchUsername(id) => {
    return UserDB.get(id)
      .then((data) => data.username)
}
```

Code

Add error handling

```
function fetchUsername(id) => {
    return UserDB.get(id).then((data) => {
        return data.username
    })
    .catch((err) => {
        logger.warning('Error catched in fetchUser',
{user_id: id, erro: err})
        throw new Error(`fetchUser failed with error id
${id}, error: ${err.message}`)
    })
}
```

Common problem: connection management

If you are making a lot of outgoing connections per request consider using pooling. Network connections are limited resource and it may become a bottleneck for your system or the system on the receiving end.

For database connectors pooling is a common thing to use, but not as common for http connectors.

Code

Request module has pool argument

```
request({
    url: s.url,
    pool: {maxSockets: 10}
})
```

Can be used to set maximum amount of connections for each external service.

System

How we run performance/stability test

- Choose a few common user journeys with some variability
- Simulate the user journeys on http level
- Mock external services if needed
- Run the journeys with test runner with enough concurrency

We use Locust (it's Python!)

```
class MyTaskSet(TaskSet):
    min_wait = 5000
    max_wait = 15000

    @task(10) # Weight of 3
    def searchRandomProduct(self):
        pass

    @task(20) # Weight of 6
    def loadDetailPage(self):
        pass

    @task(1) # Weight of 1
    def logout(self):
        pass
```

How we run performance/stability test

Run two tests:

Max throughput:

Find the limits of the system, analyze if the system handles it gracefully.

Long term stress:

Run a test with less load but leave it running 10+ hours.

Security

There are two conflicting ideologies on using libraries:

- “I don’t need to use library-x, it’s too heavy and I just need a few of its features. I’ll just write it from scratch”

- “I’ll just leverage existing library for this, why would I spend time writing something someone else already wrote?”

Security

When it comes to anything related to security, avoid writing any code yourself! Especially if it’s about encryption or authentication.

Security

There are a bunch of common security related threats one that is fairly easy to fix is to use the correct HTTP headers

There are a lot of small details related to request and response http headers that can affect the security of the system. Helmet is a good package to give some of the security with minimal effort:

```
app.use(helmet())
```

It will adjust the headers of your application for you.

You don’t have to solve everything with code!

People tend to solve problems with tools that they are familiar with. For programmers this means solving problems by writing code.

However, there may sometimes be an alternative approach.

You don’t have to solve everything with code!

Example 1: Caching

Implementing caching logic in your code easily increases its complexity and also introduces un-determinism to the system (makes testing more difficult).

There are powerful caching solutions (like Varnish) that can be integrated without modifying any of the code. It may be less flexible but for production systems keeping complexity lower is a huge benefit.

You don’t have to solve everything with code!

Example 2: Connection limiting

We talked earlier about limiting the amount of connections to a server. Doing this with code is not the only option.

Load Balancers can be introduced between two services to limit the number of allowed connections and buffer the incoming requests. This can help to protect the upstream server from too many requests.

You don’t have to solve everything with code!

Talk to your infrastructure/devops guys and be creative when solving problems.

If you can solve the problem (even partially) using just infrastructure changes then it may be the best choice for you, since you won’t have to write any code and the complexity of your API does not grow.

Sum up

Thinking

- Consider different aspects of going to production
- Be aware of missed responsibilities

Code

- Do proper logging
- Handling service failures
- Log error context
- Manage connections

Sum up

System

- Do performance / stability testing
- Don’t implement security yourself
- Leverage infrastructure level solutions

Thank you!