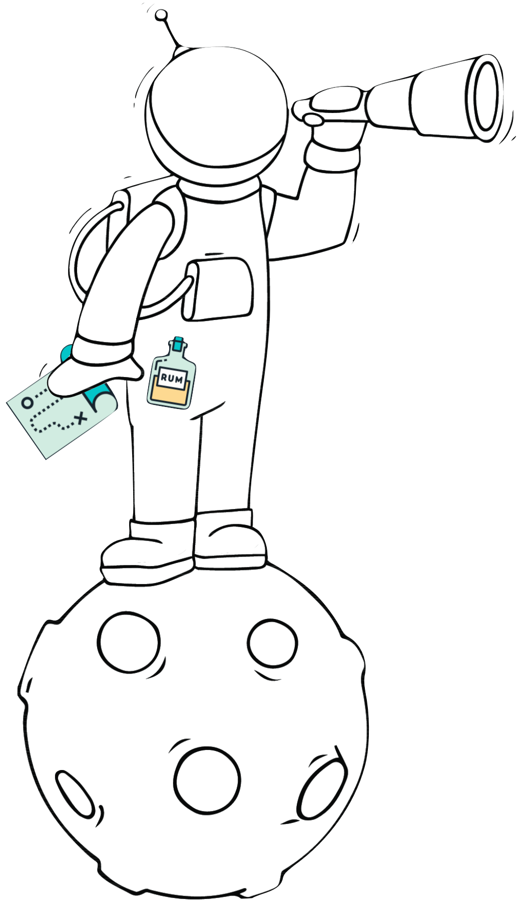


Implementing Microservice Discovery in 100 Lines of Node.js

Anup Bishnoi
@asyncanup

Disclaimer

I do work at Netflix

Disclaimer

but

Disclaimer

this talk has nothing to do with that

Disclaimer

My own,
not Netflix

Let's learn you some

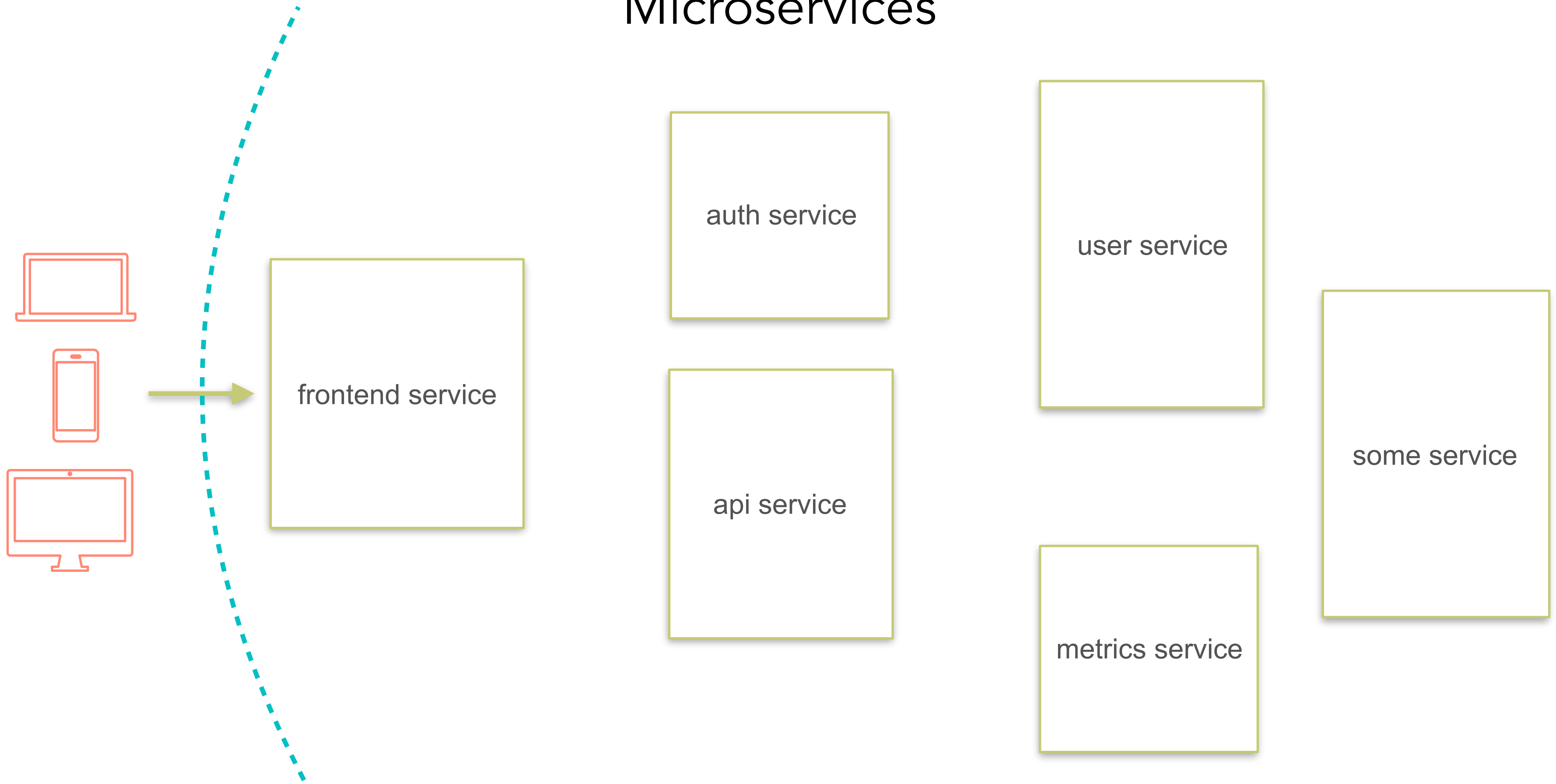
Service Discovery

for great good!

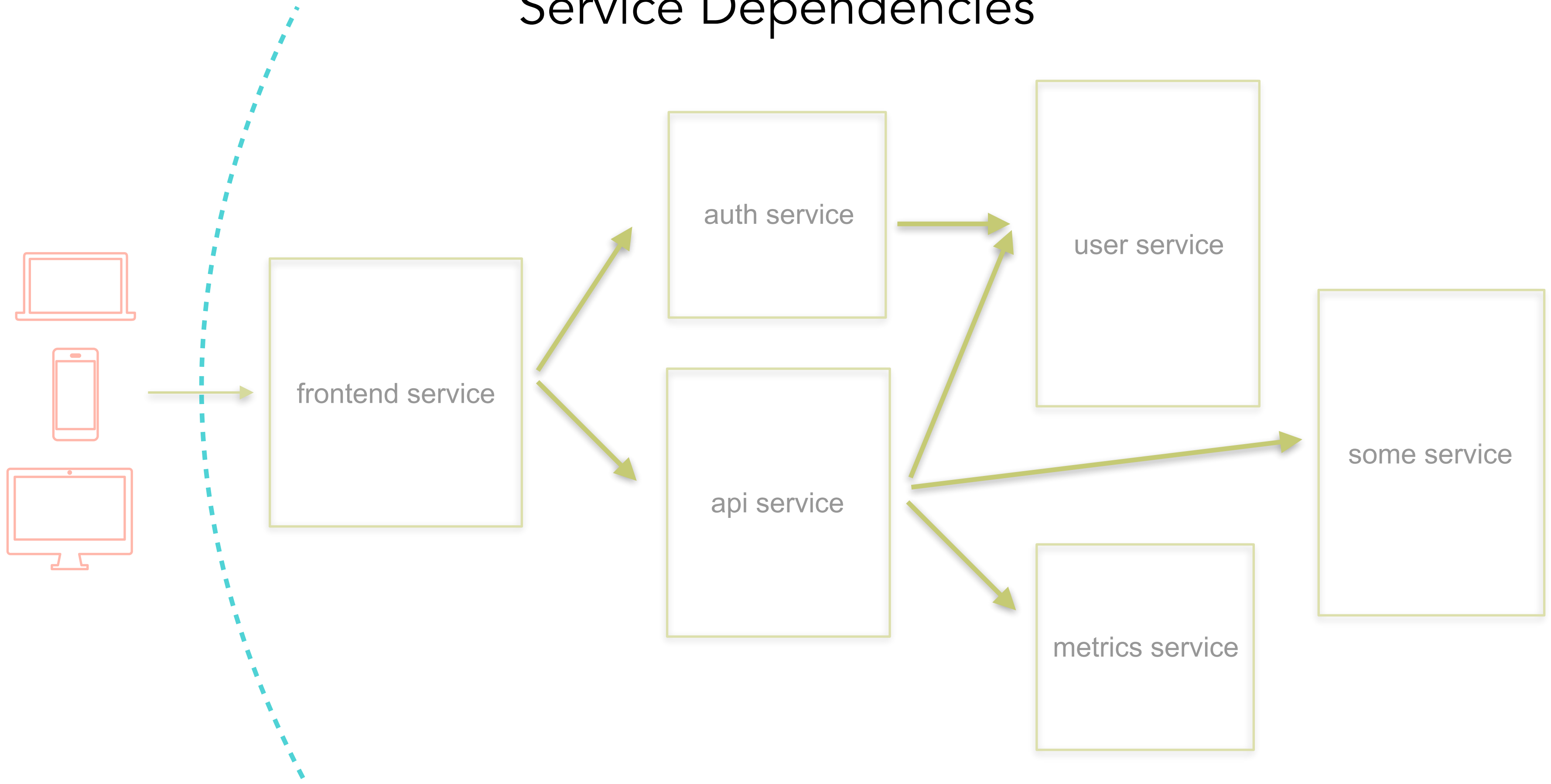
Buzz-word Alert!

Microservices!

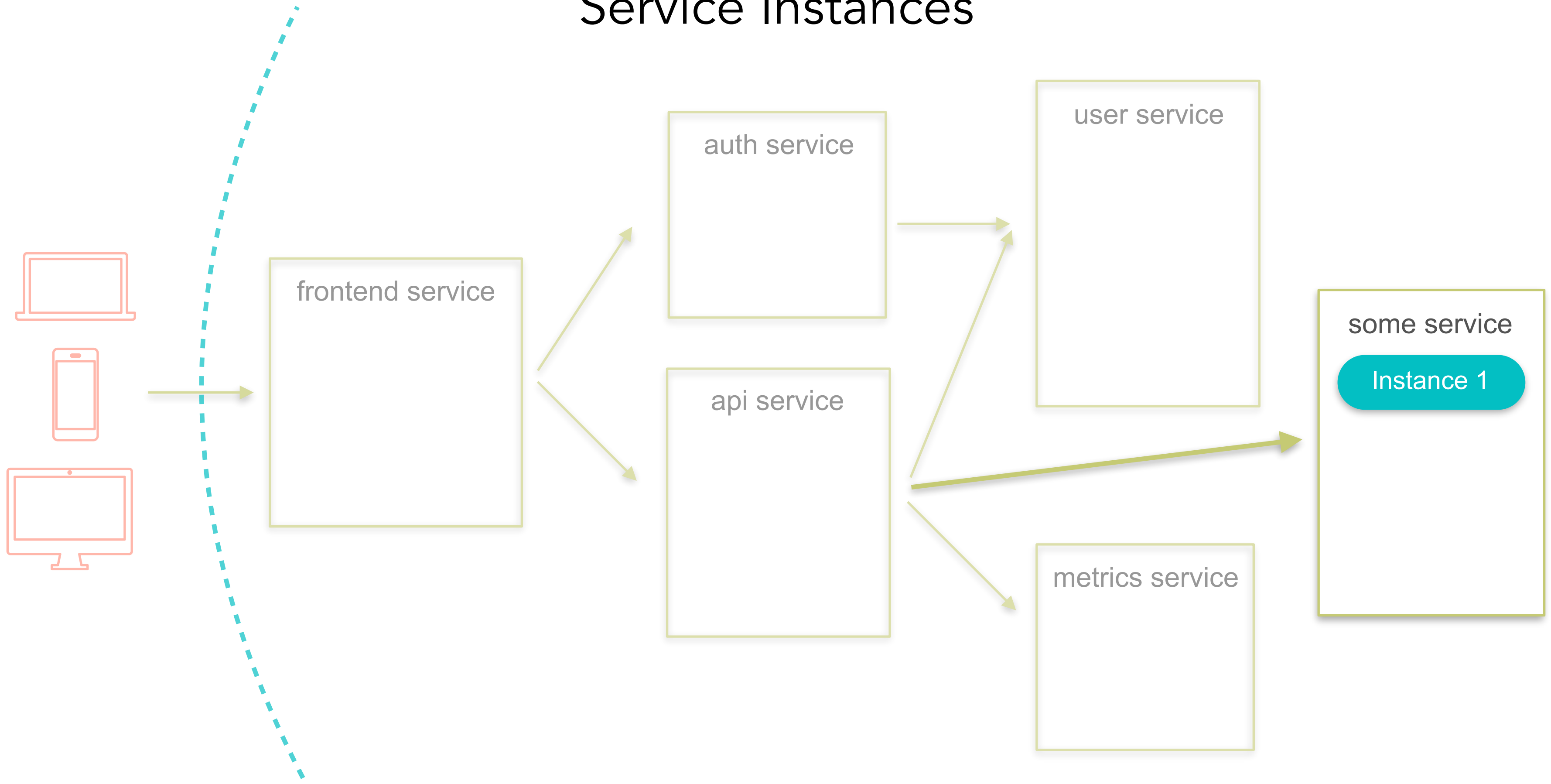
Microservices



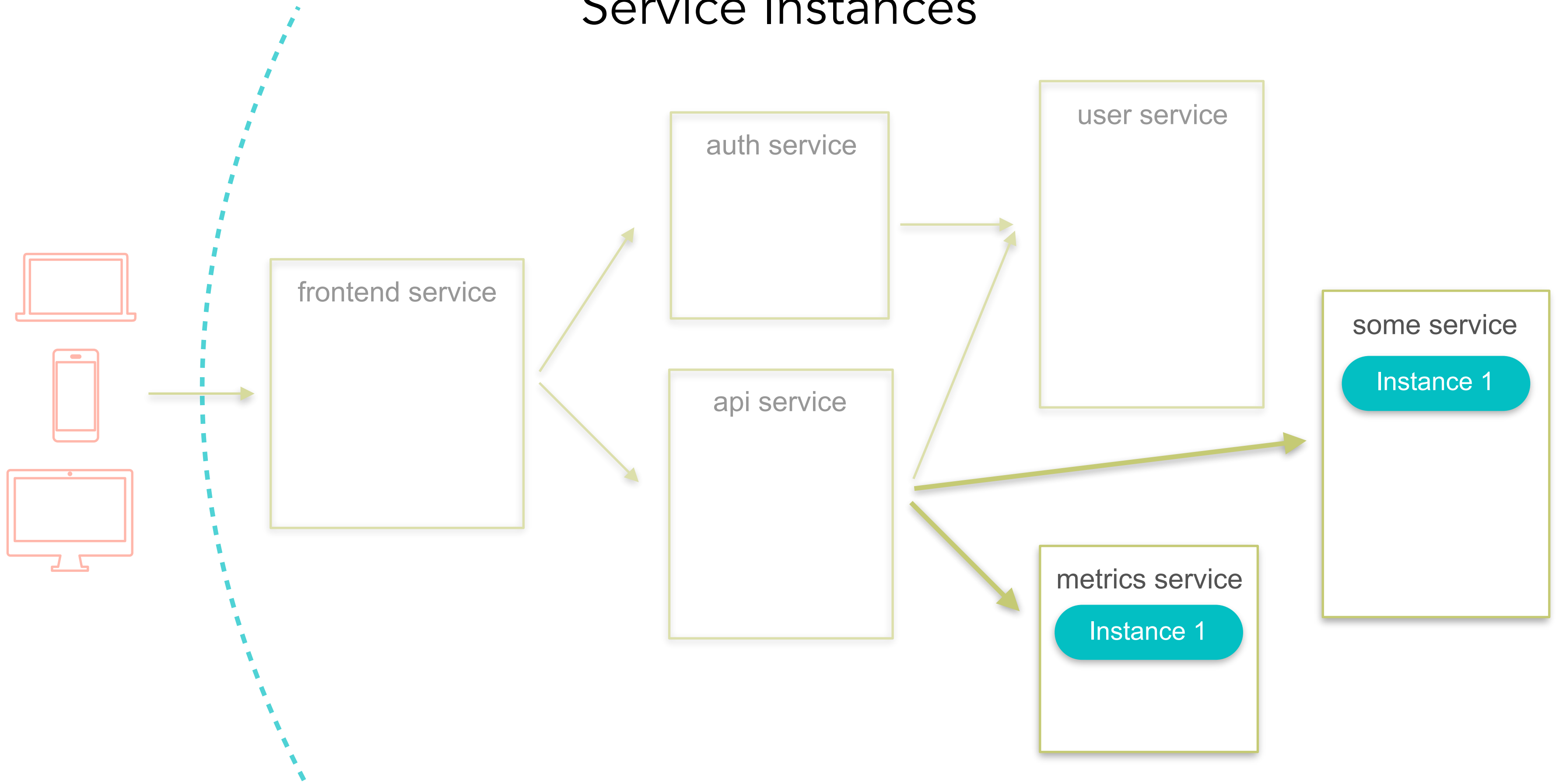
Service Dependencies



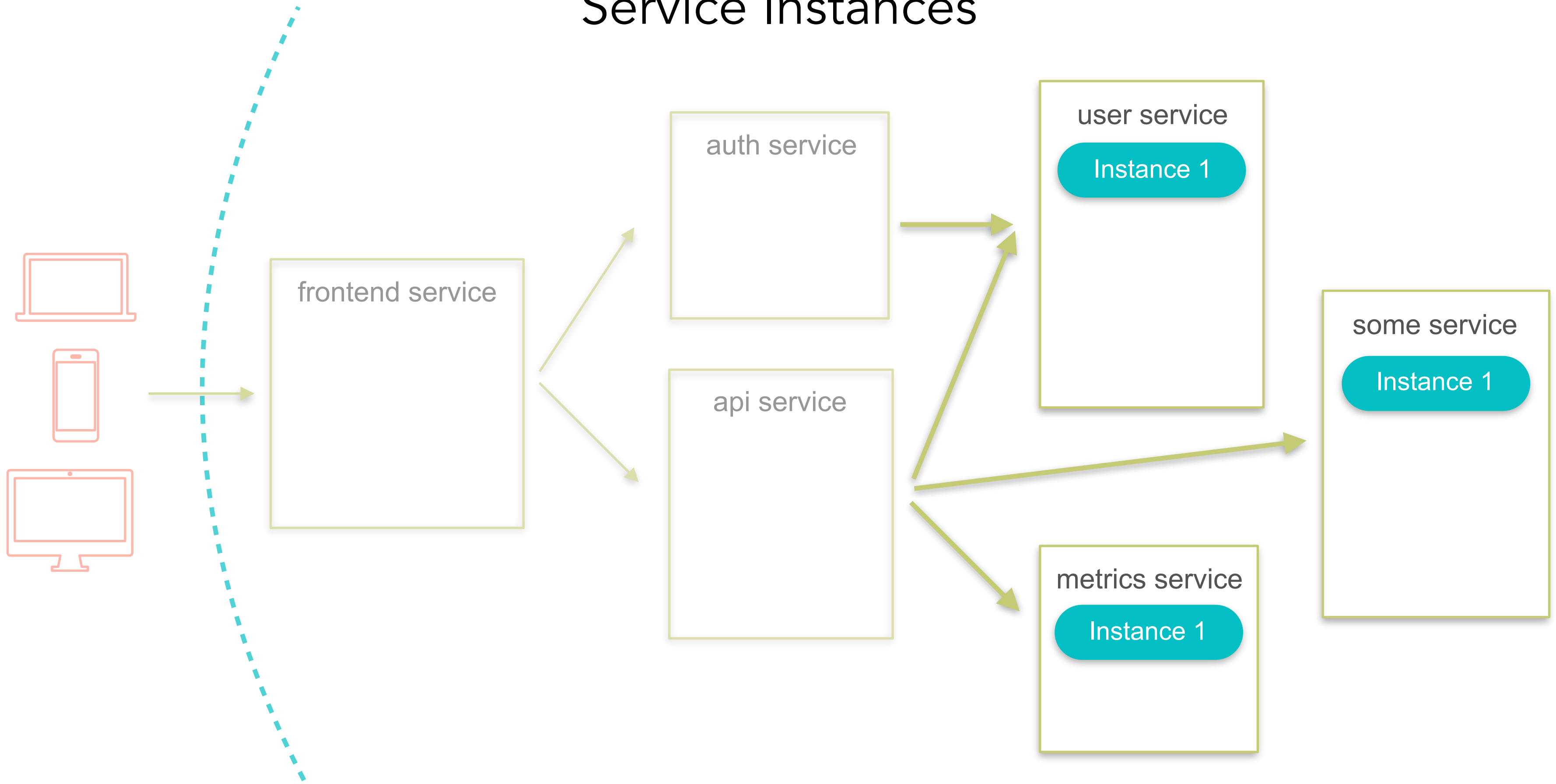
Service Instances



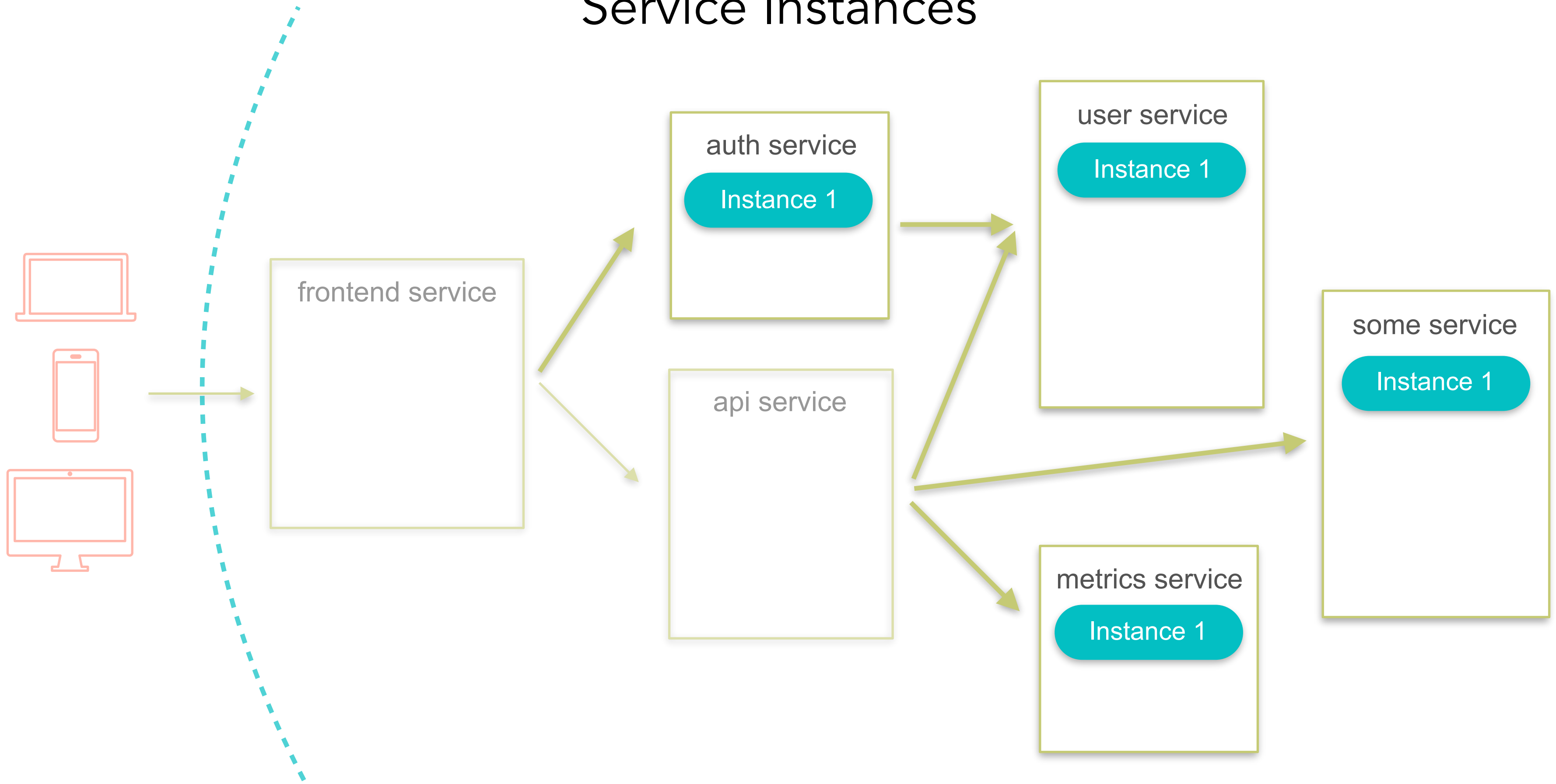
Service Instances



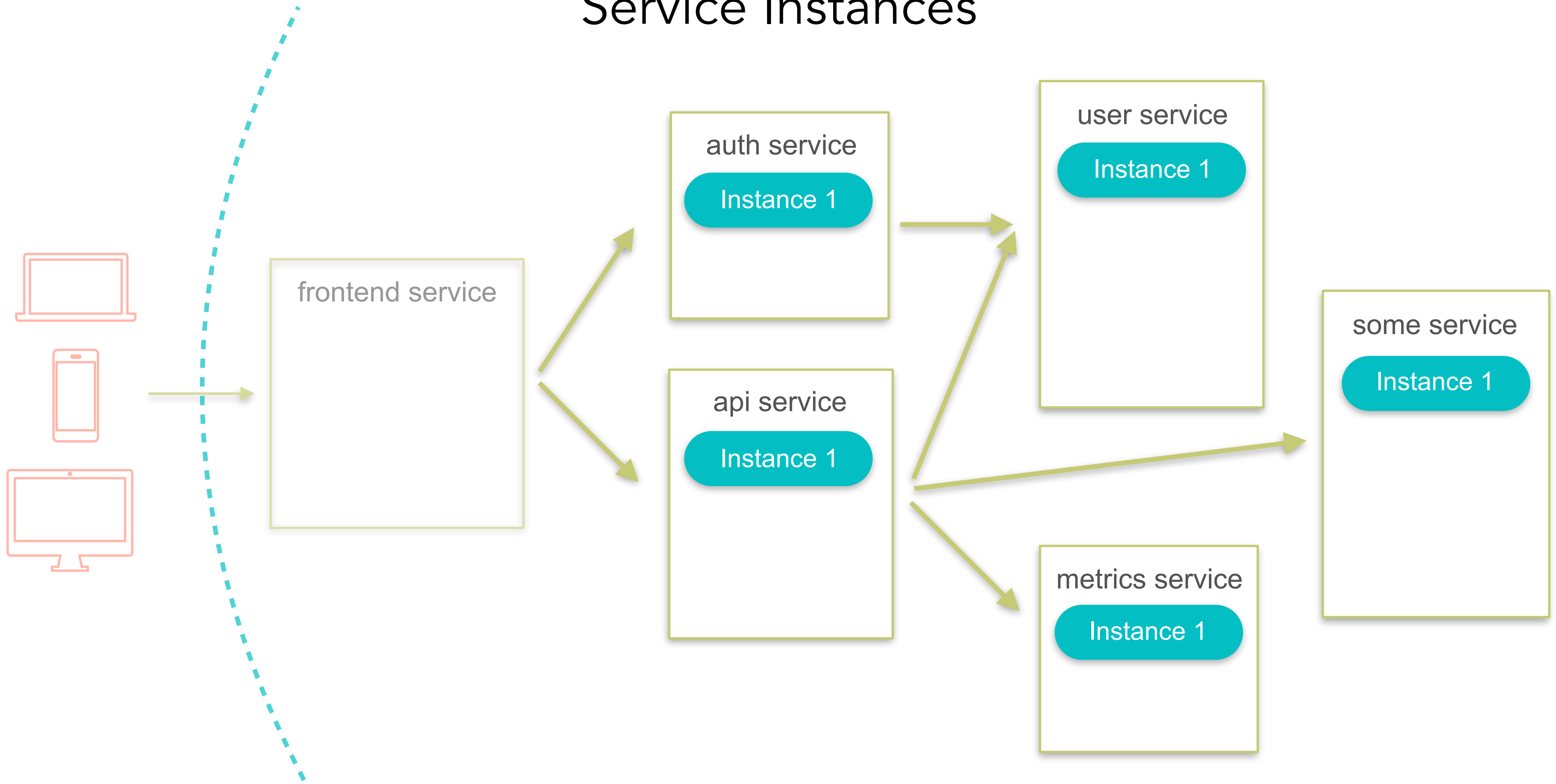
Service Instances



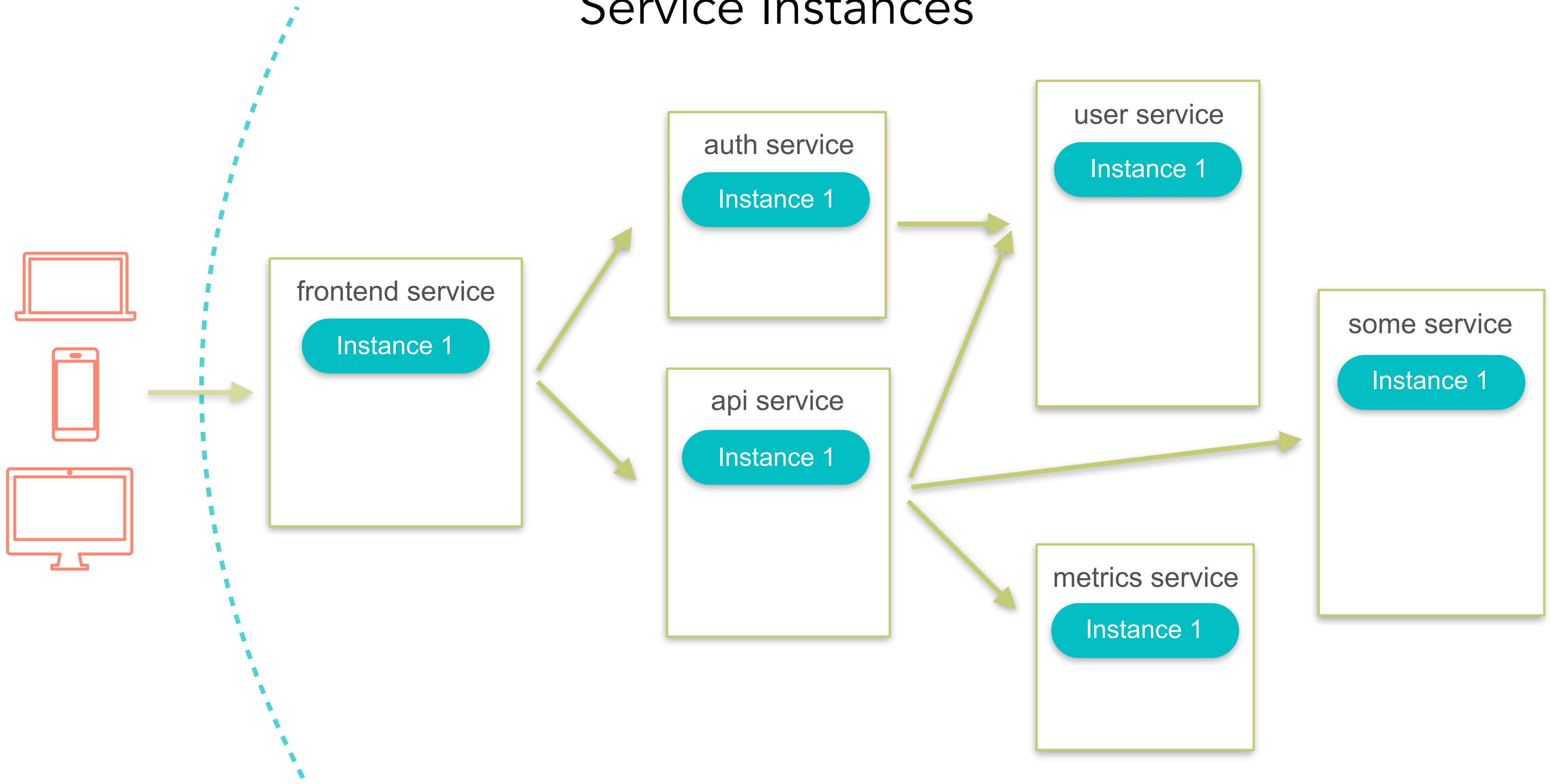
Service Instances



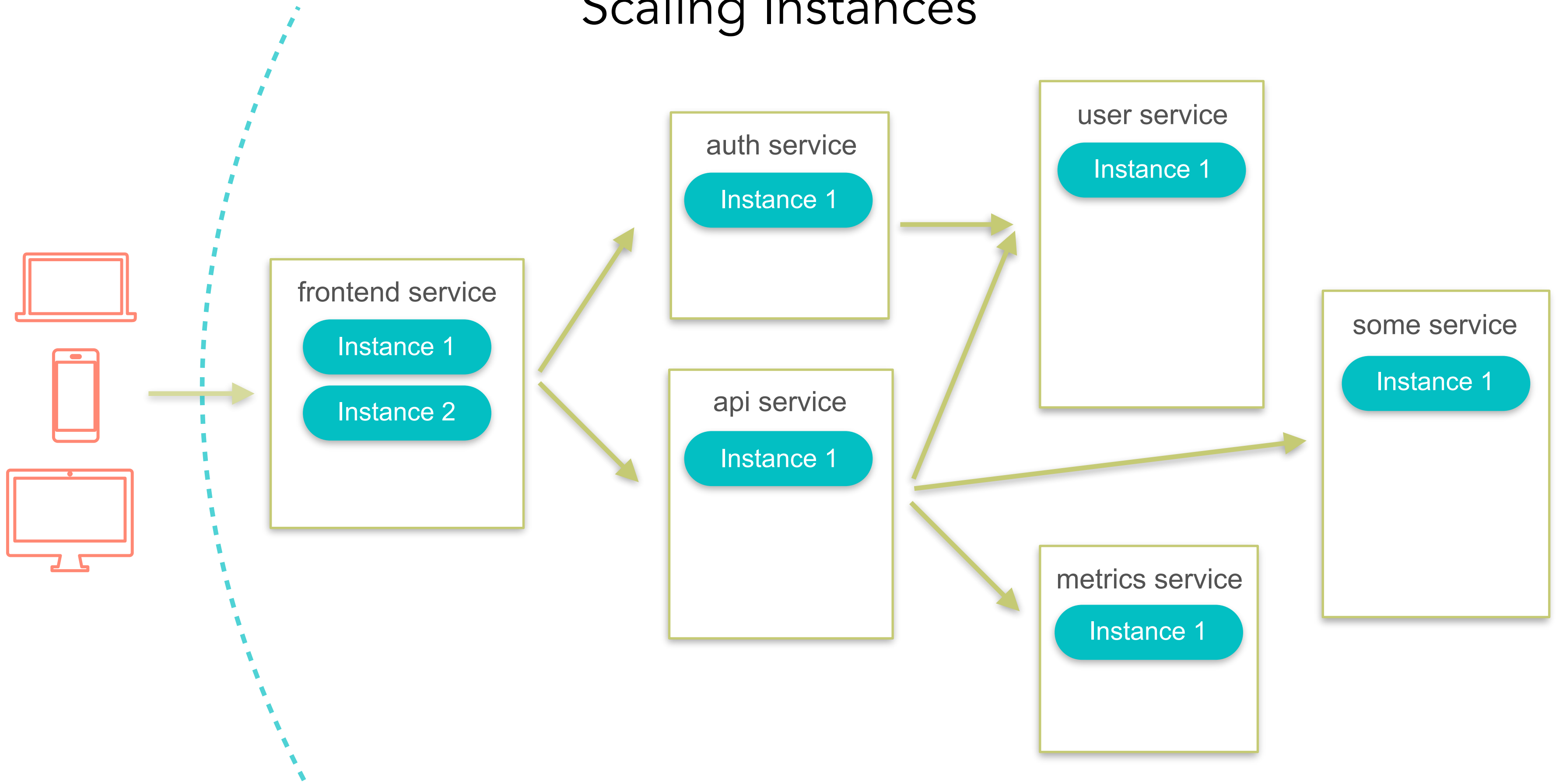
Service Instances



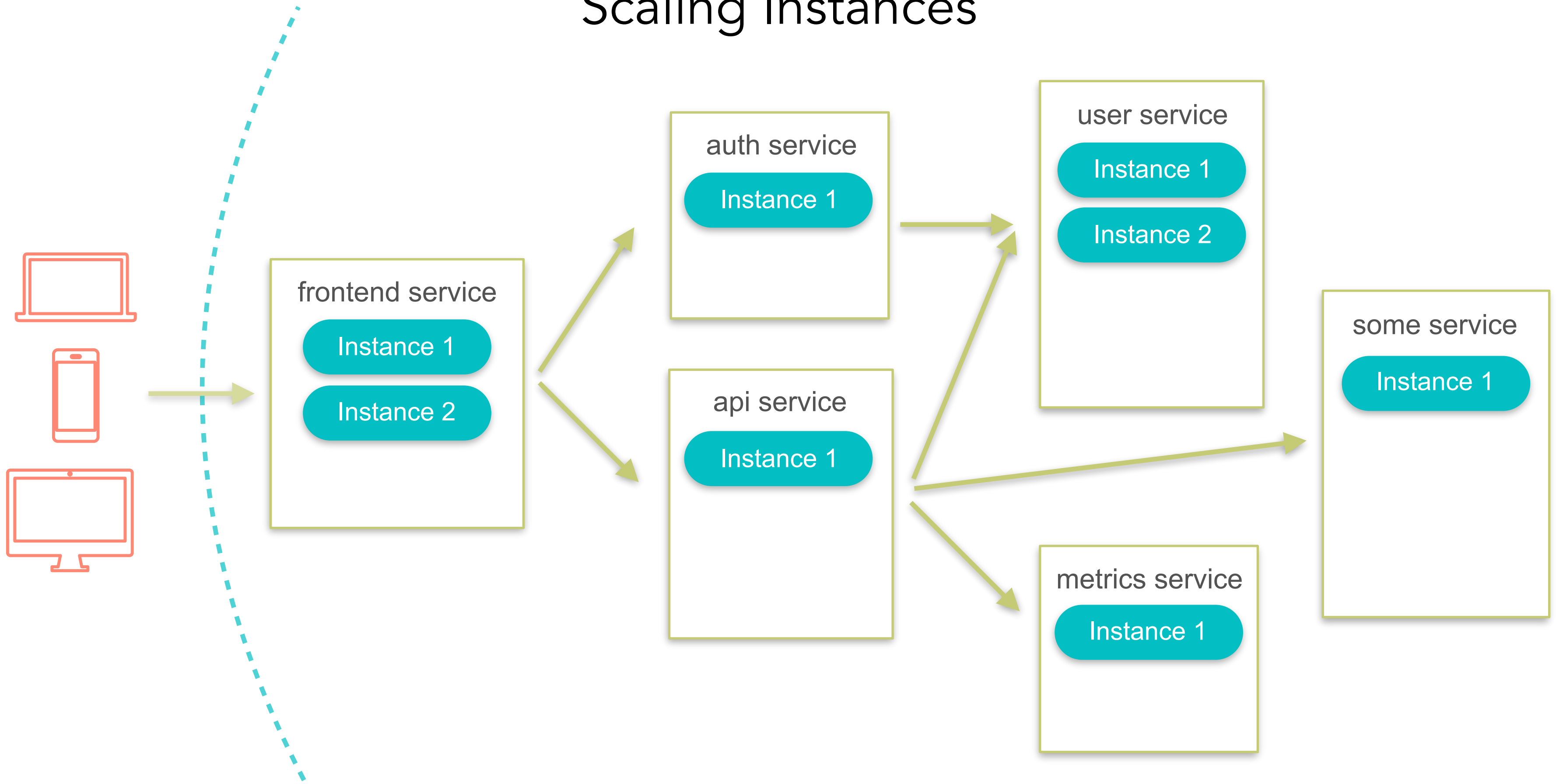
Service Instances



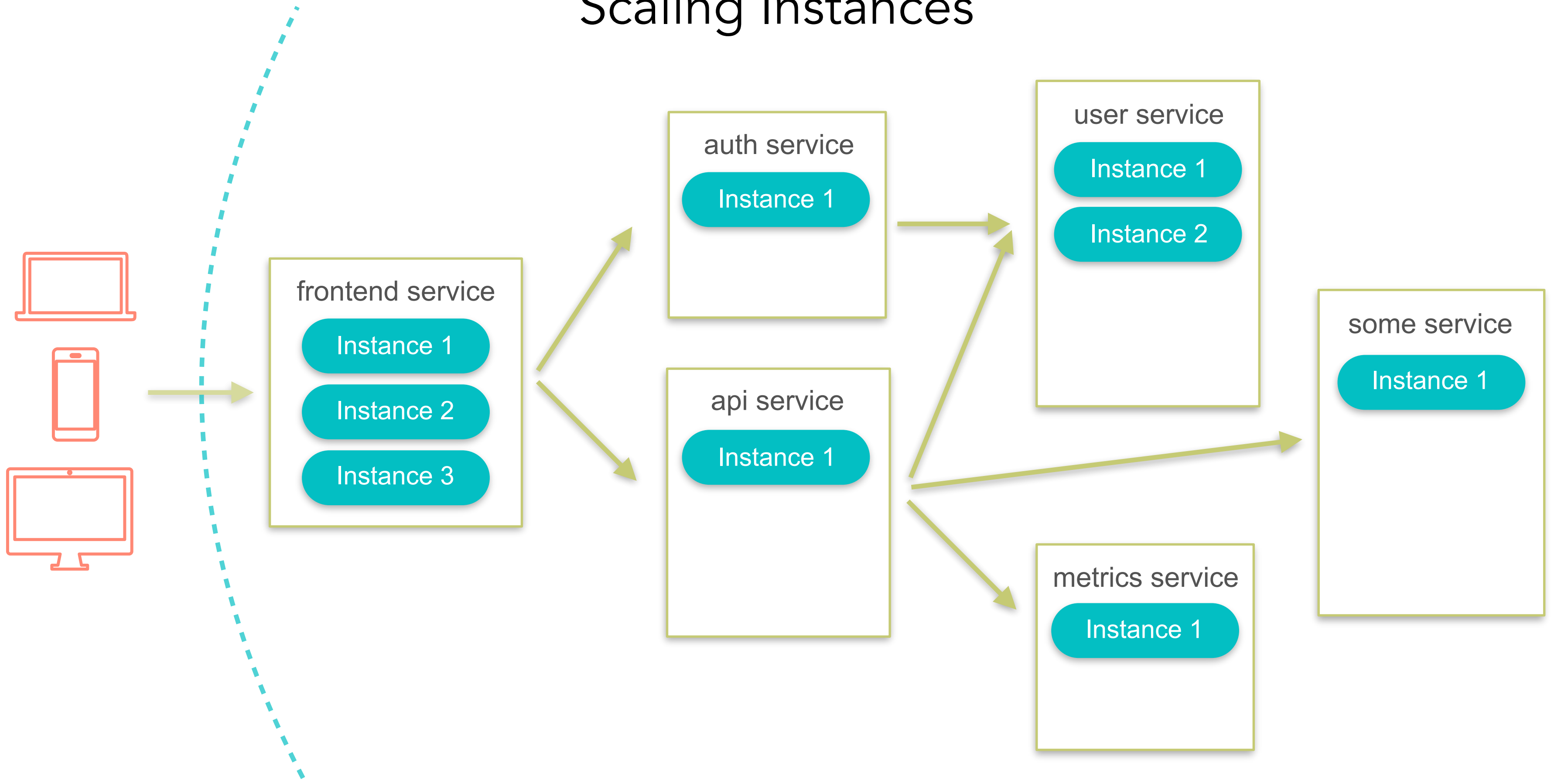
Scaling Instances



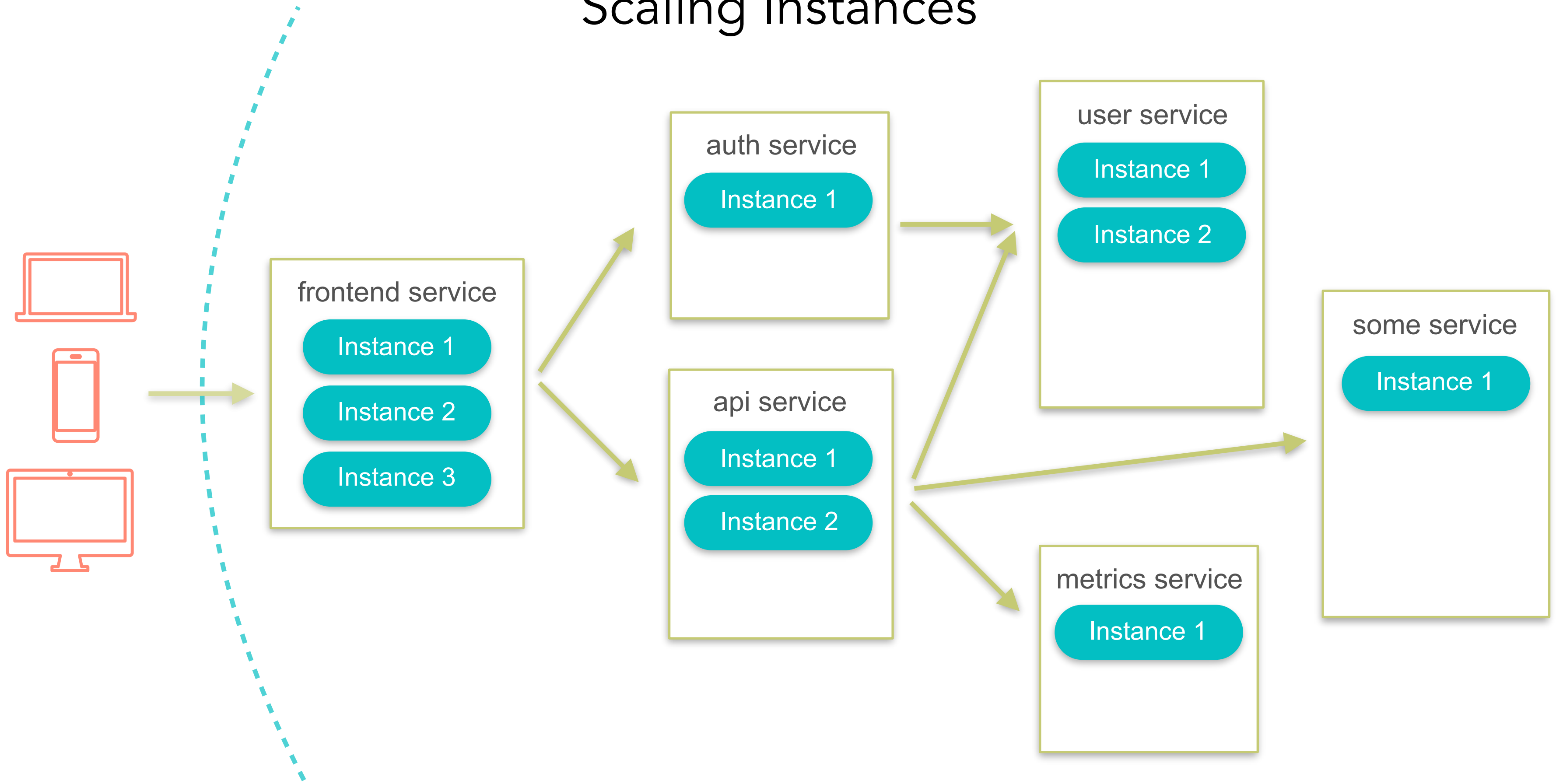
Scaling Instances



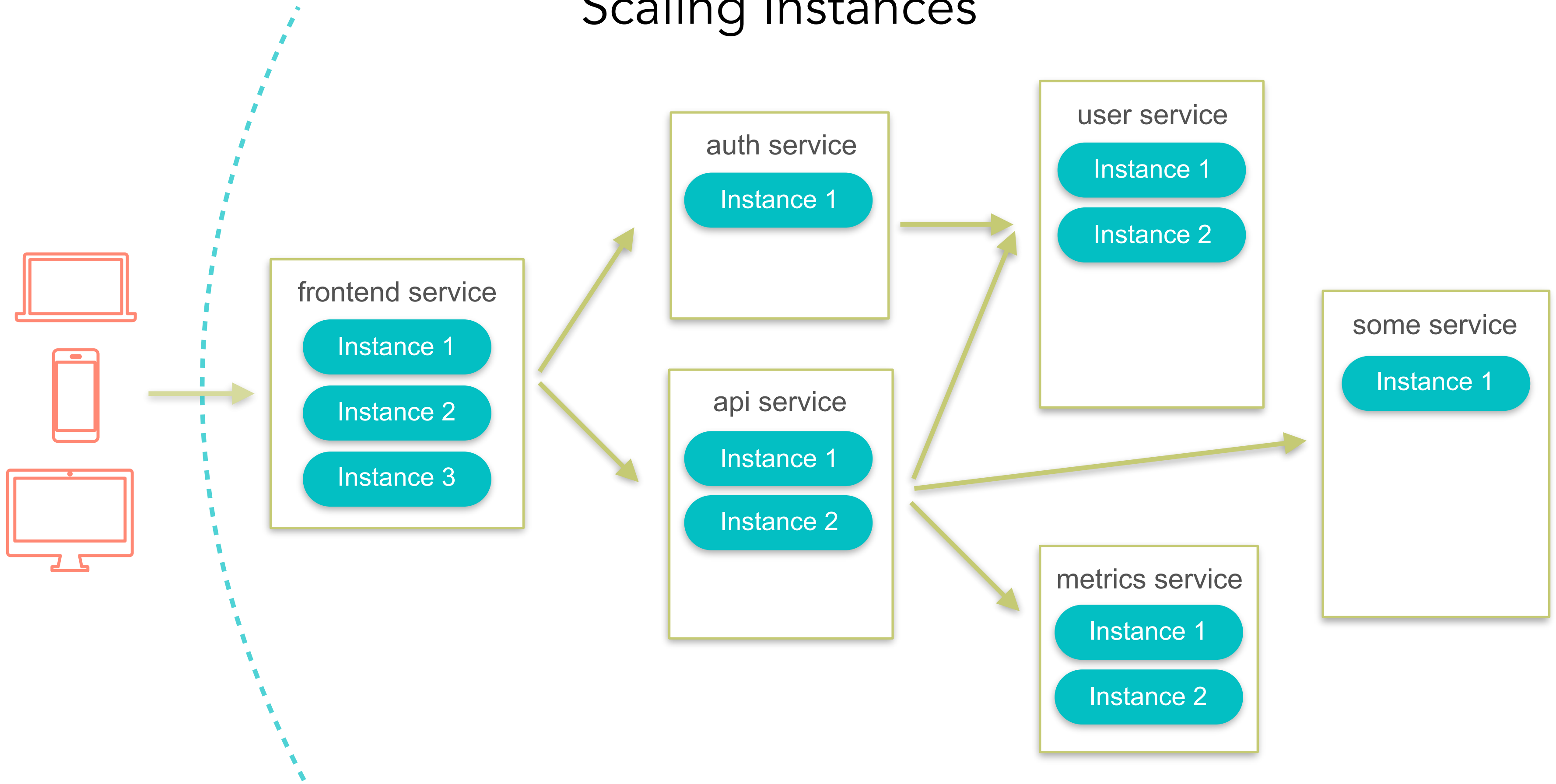
Scaling Instances



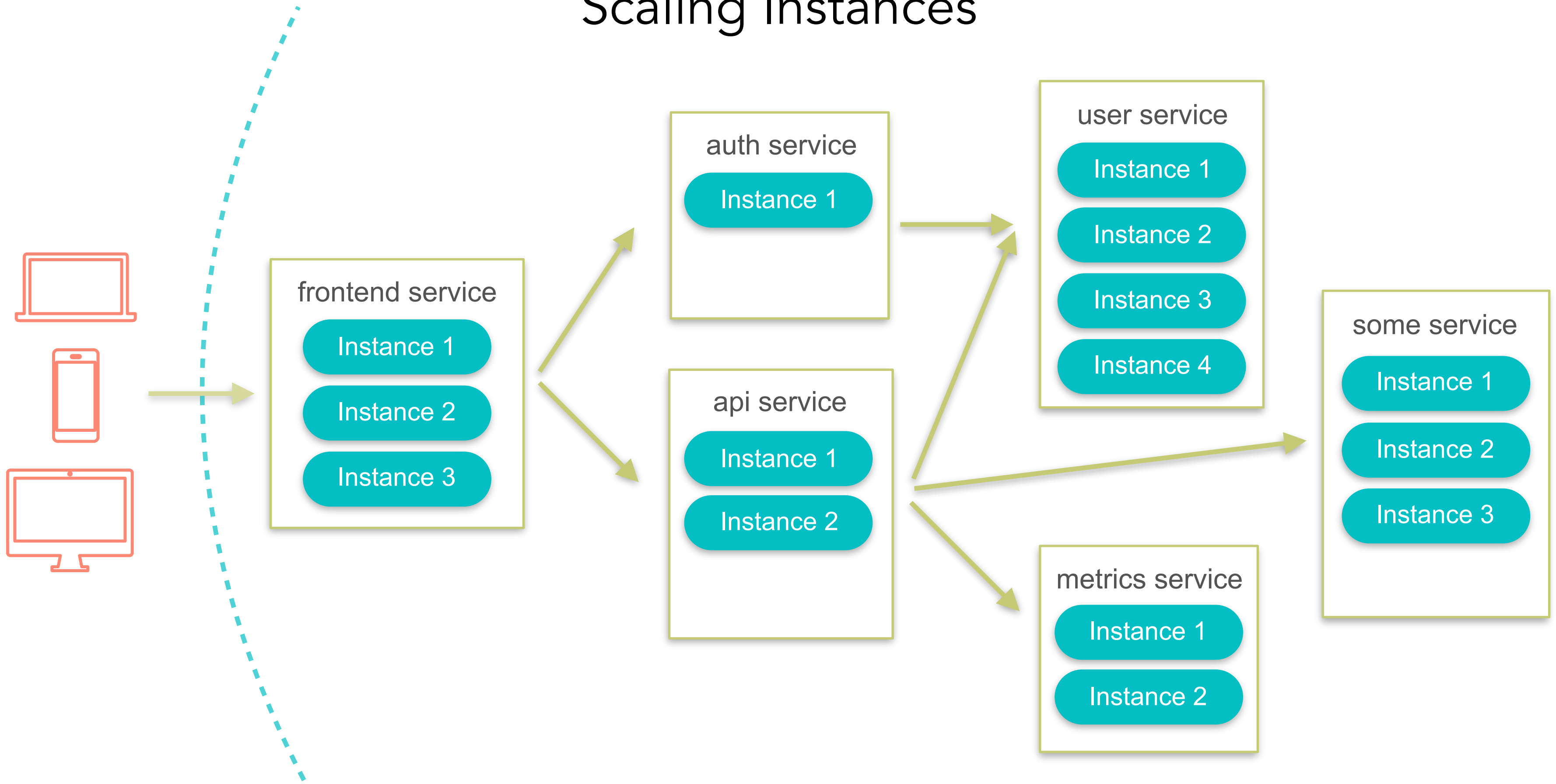
Scaling Instances



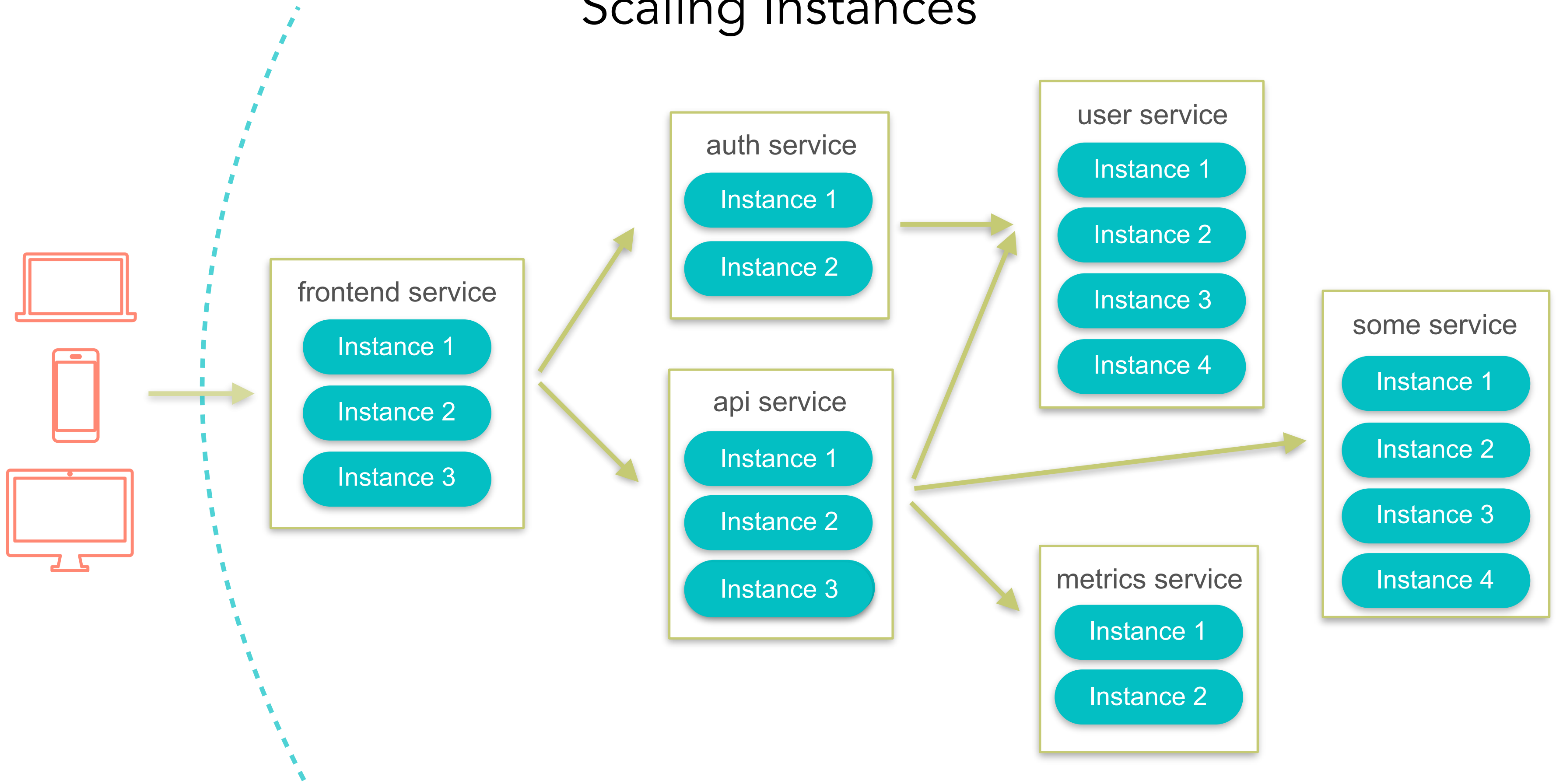
Scaling Instances



Scaling Instances



Scaling Instances



Microservices must:

Deploy Independently

Microservices must:

Scale Independently

Microservices must:

Survive Individual Failure

To do their job,
Microservices need…

SERVICE
DISCOVERY

Microservices need Service Discovery

Deploy Independently

Service Registry

Microservices need Service Discovery

Scale Independently

Load Balancing

Microservices need Service Discovery

Survive Individual Failure

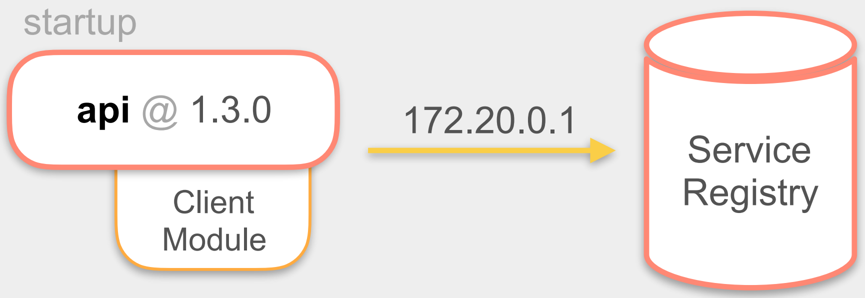
Live Health Checks

How does Service Discovery work?

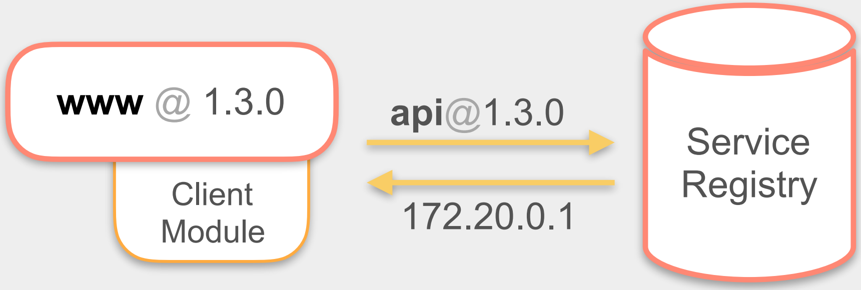
Registration

Discovery

Self-Registration



Client-side Discovery



introducing...

vasco

The Great Service Discoverer

Setup

Vasco
Registry

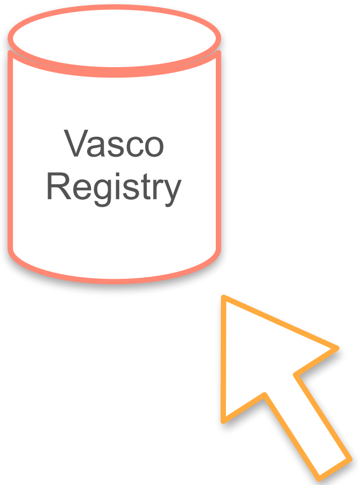
Setup

Vasco
Registry

Redis Database

Fast Key-Value Store

Setup

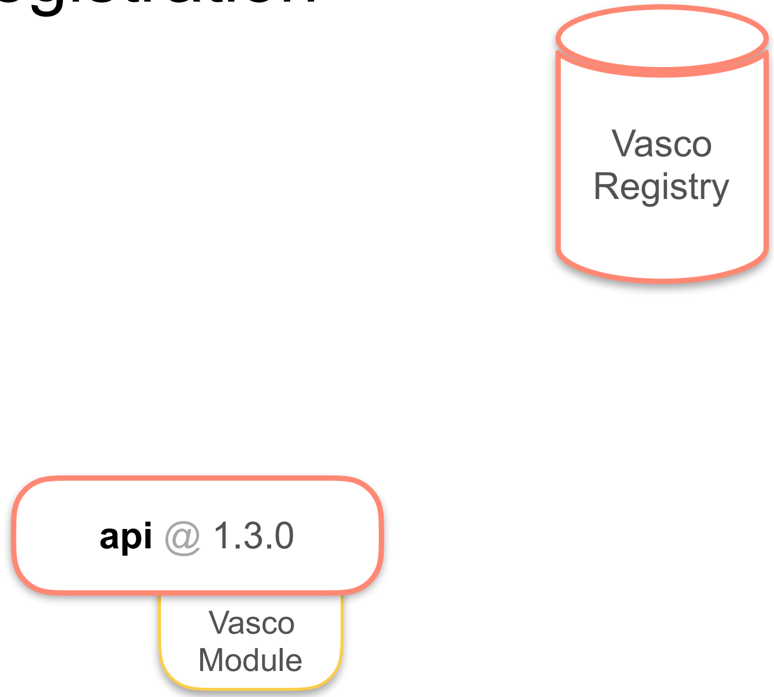


Redis Database

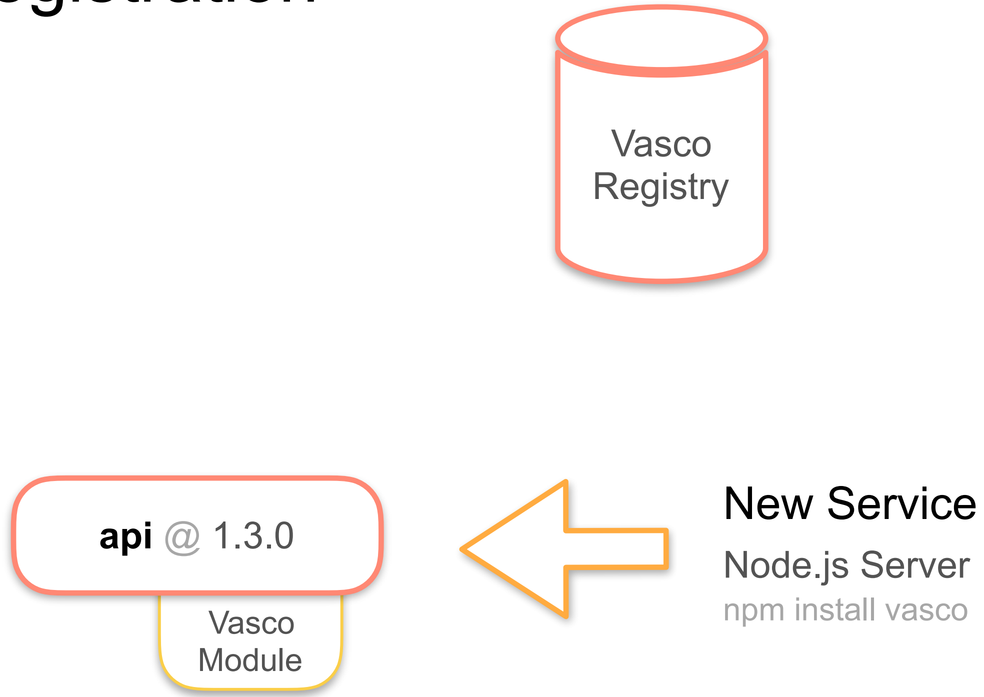
Fast Key-Value Store

Available via Amazon Elasticache

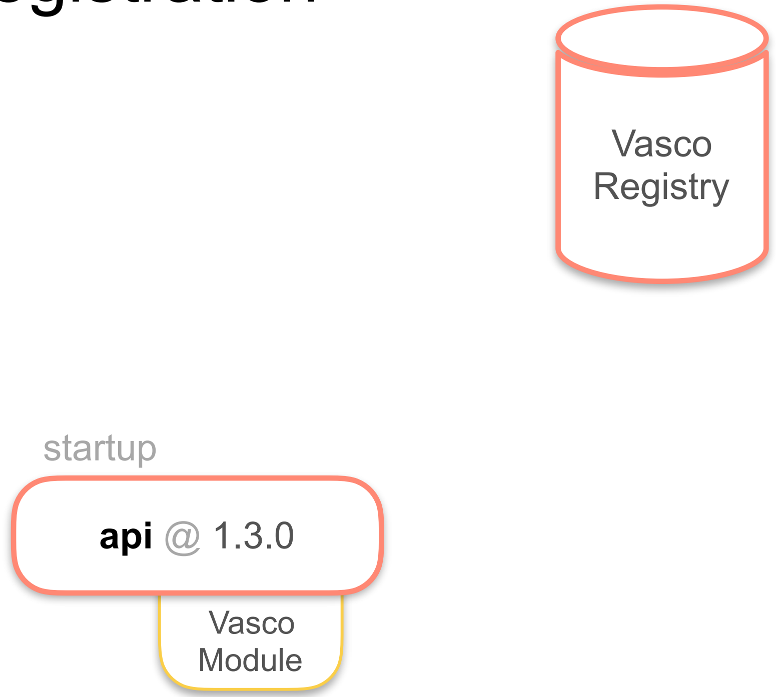
Registration



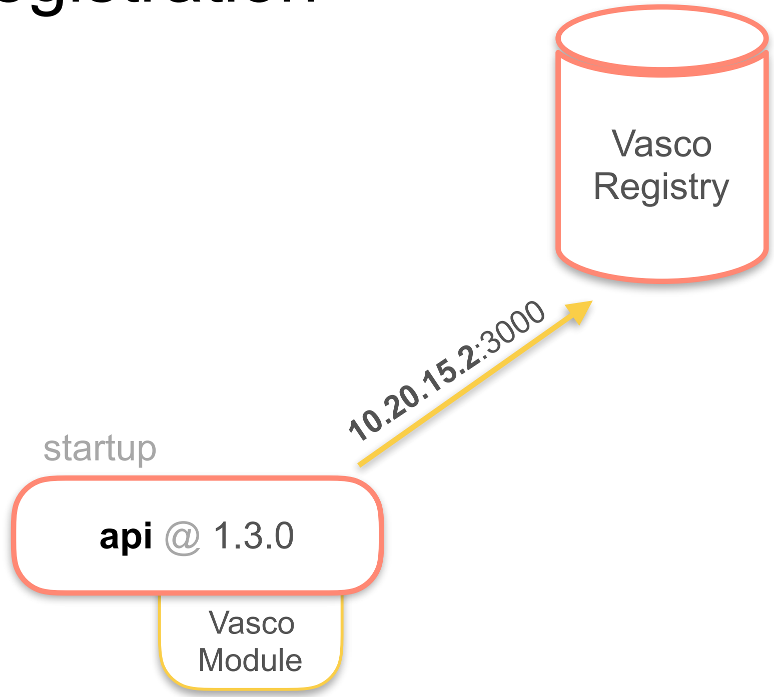
Registration



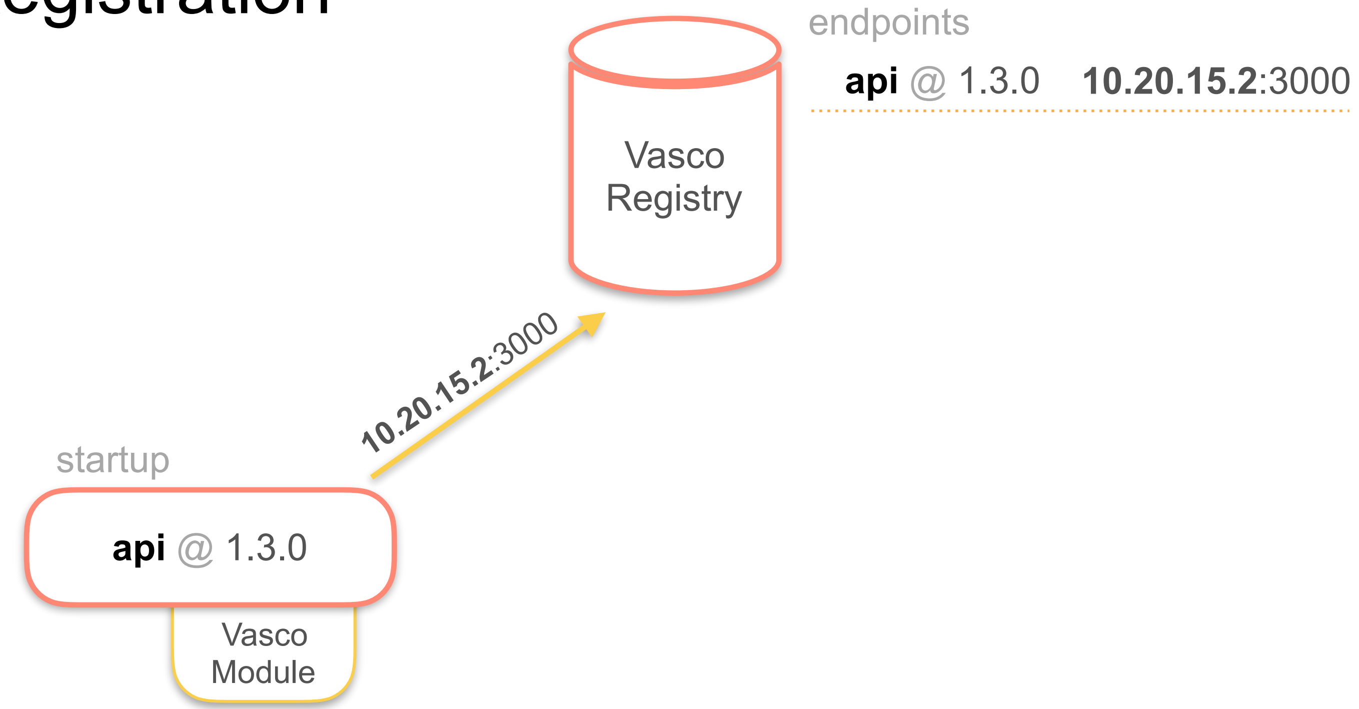
Registration



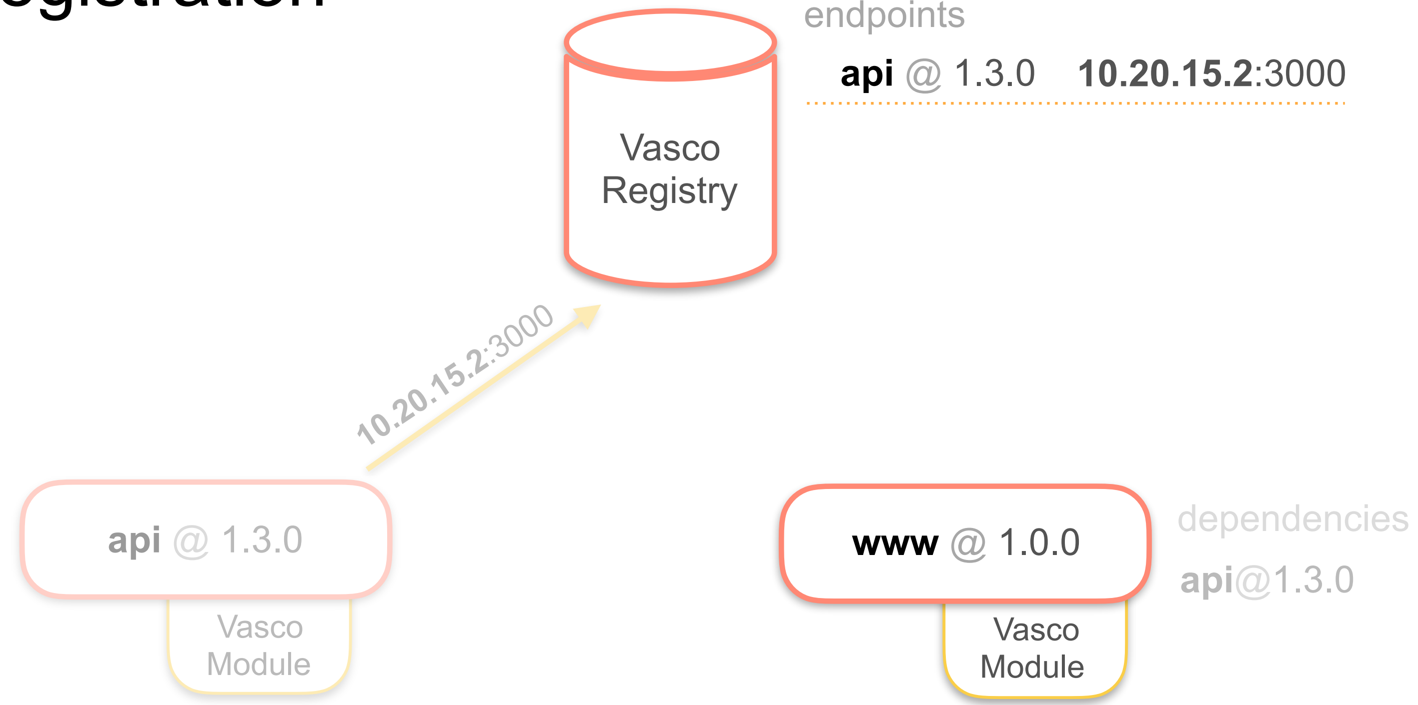
Registration



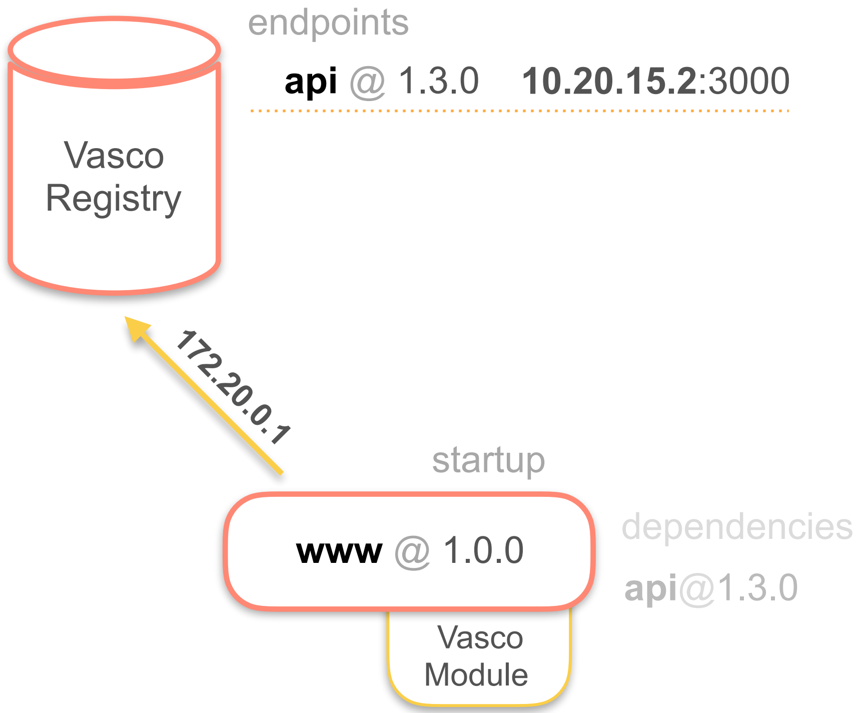
Registration



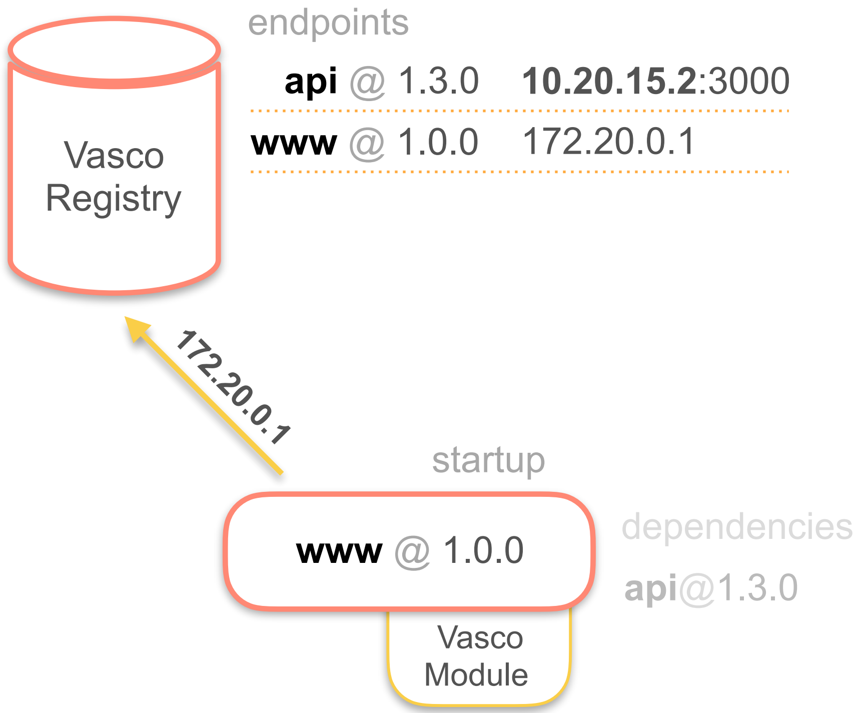
Registration



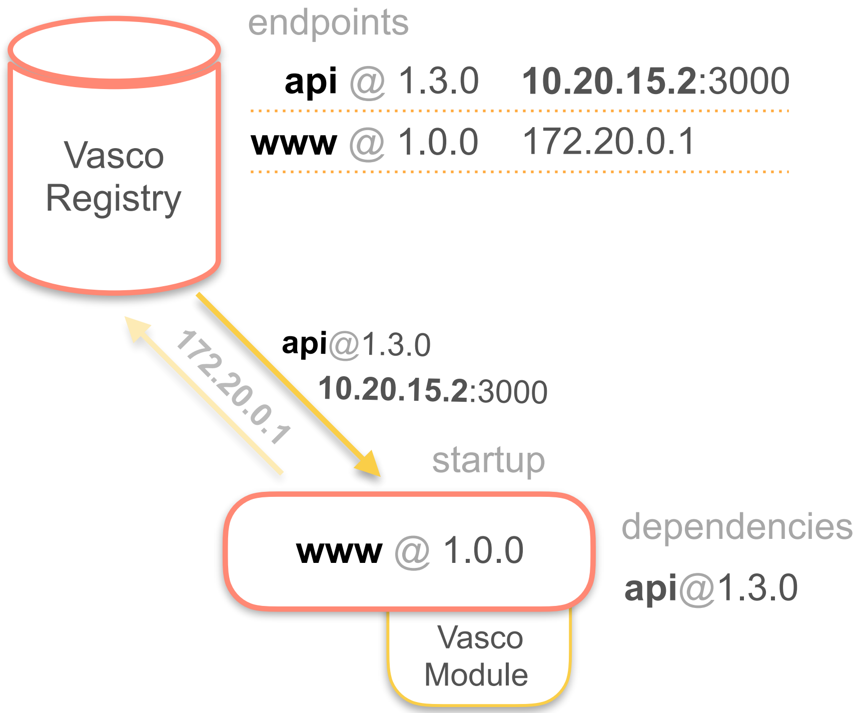
Registration



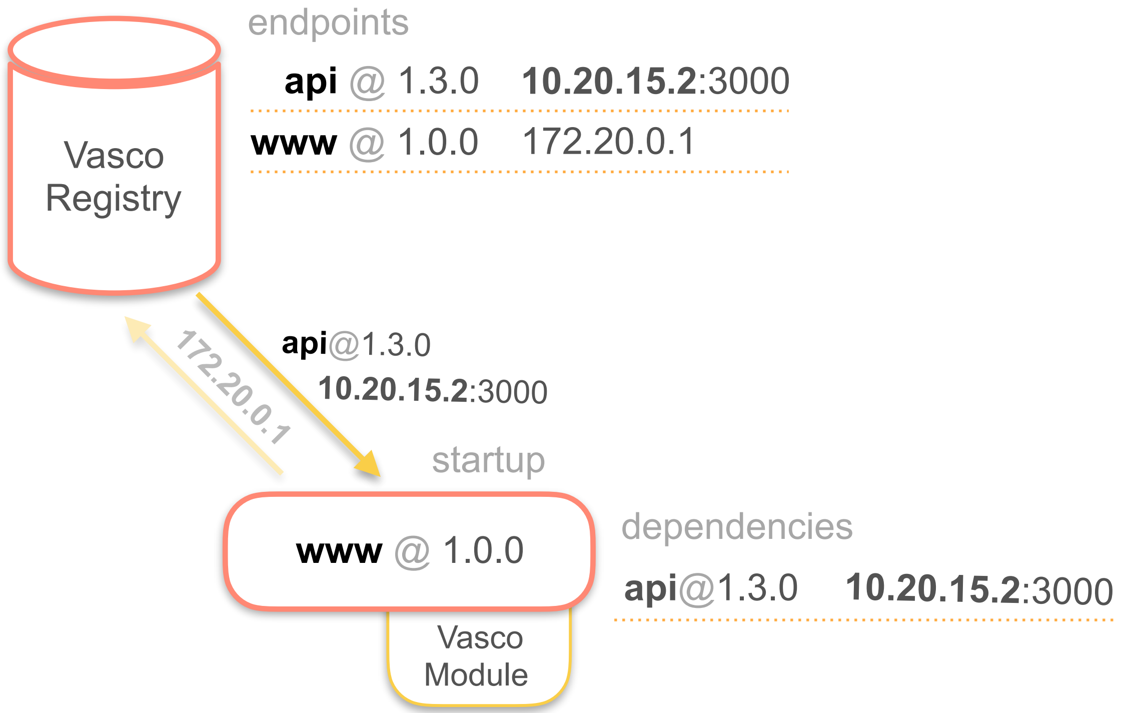
Registration



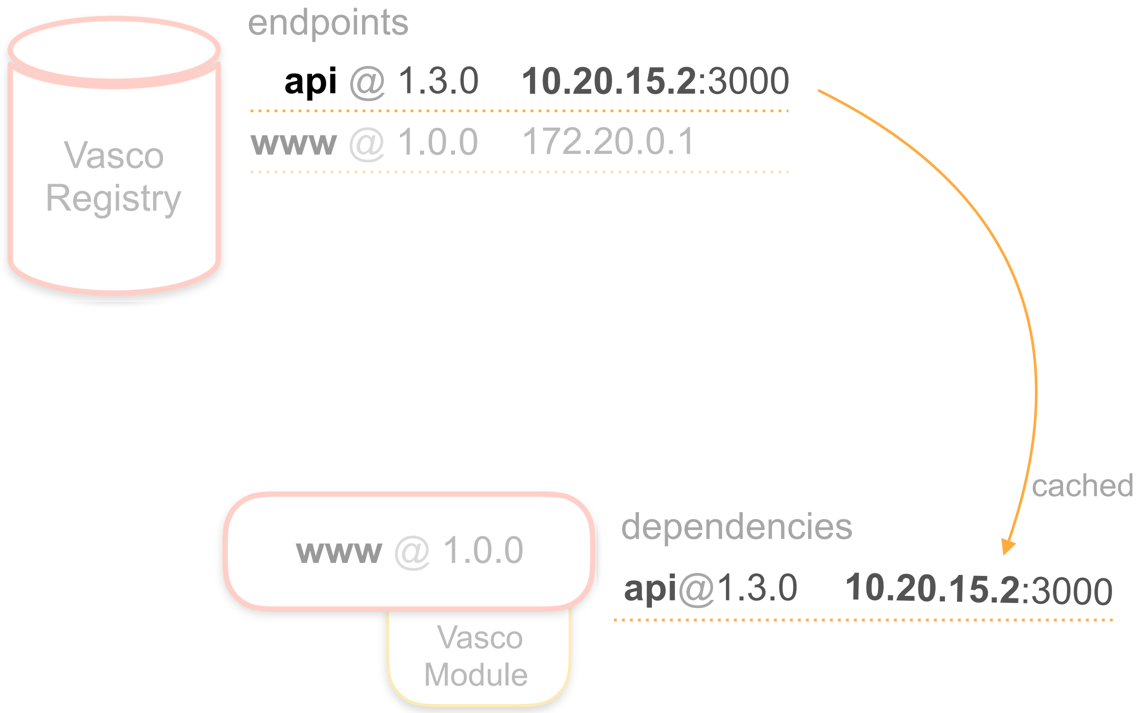
Discovery


endpoints
api @ 1.3.0 **10.20.15.2**:3000
www @ 1.0.0 172.20.0.1
Vasco Registry
172.20.0.1
api@1.3.0
10.20.15.2:3000
startup
www @ 1.0.0
Vasco Module
dependencies
api@1.3.0

Discovery

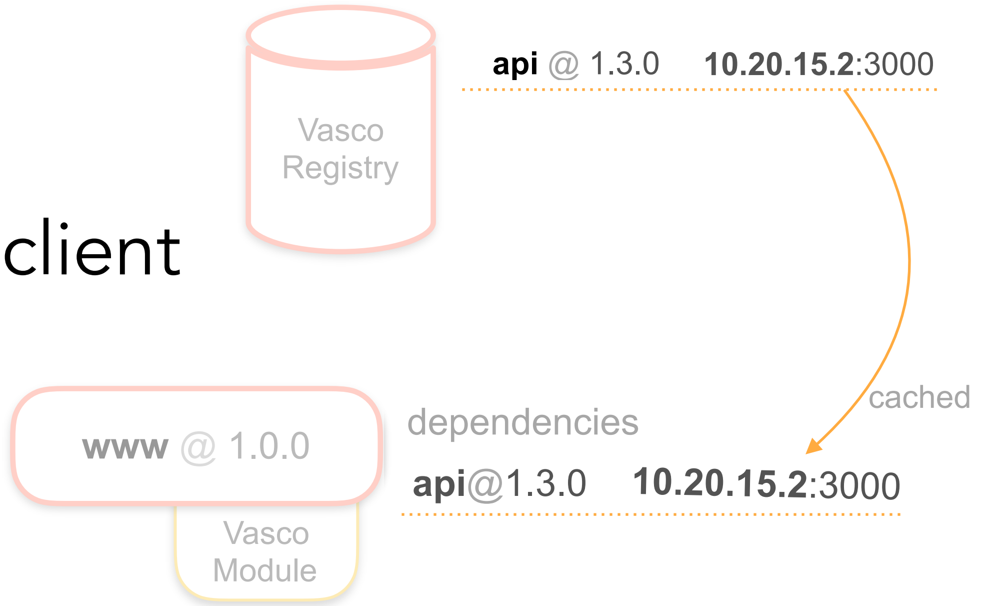

endpoints
api @ 1.3.0 10.20.15.2:3000
www @ 1.0.0 172.20.0.1
Vasco Registry
172.20.0.1
api@1.3.0
10.20.15.2:3000
startup
www @ 1.0.0
Vasco Module
dependencies
api@1.3.0 10.20.15.2:3000

Discovery


endpoints
api @ 1.3.0 10.20.15.2:3000
www @ 1.0.0 172.20.0.1
Vasco Registry
cached
dependencies
api@1.3.0 10.20.15.2:3000
www @ 1.0.0
Vasco Module

Feature: Client-side Discovery

Fetch and cache dependencies on the client service.

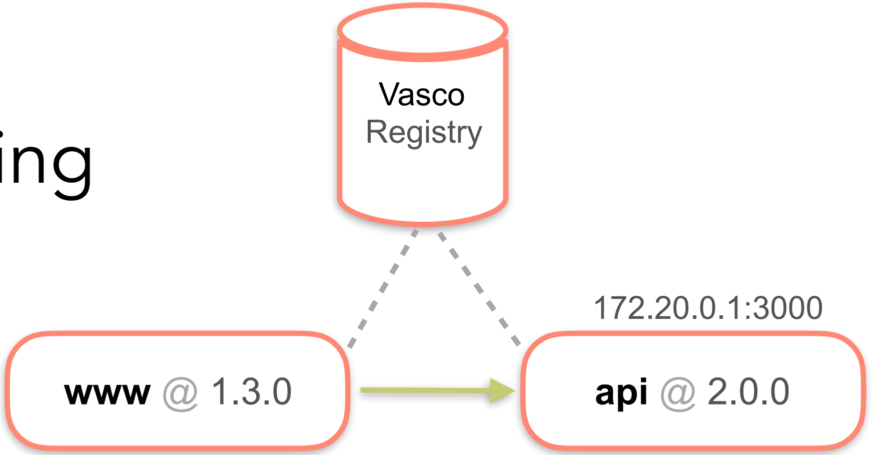


Feature: Client-side Discovery

Re-fetch dependencies if dependencies fail.

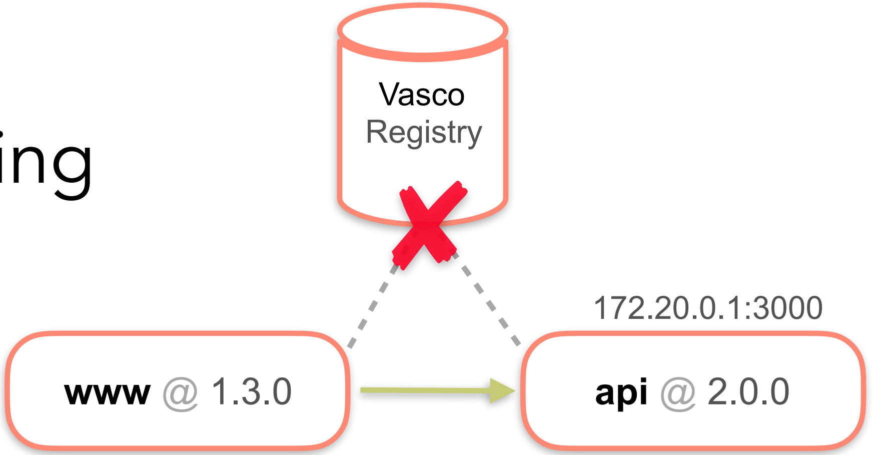
Feature: Client-side Discovery

Vasco failure does not bring down services.



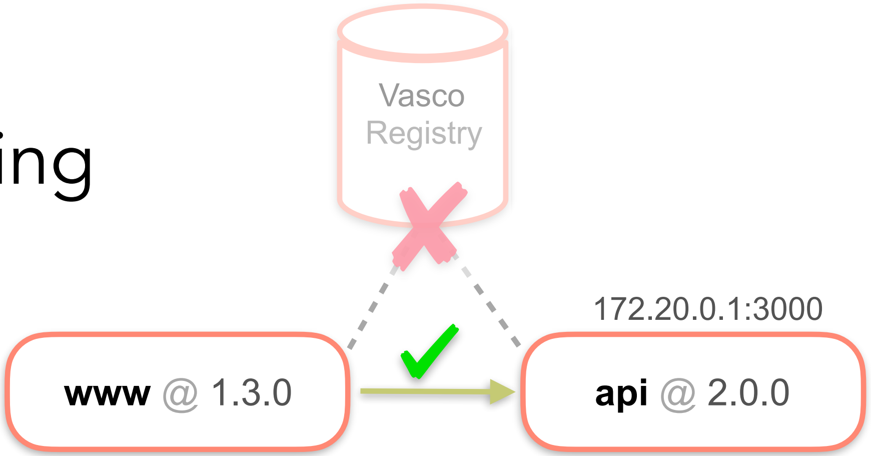
Feature: Client-side Discovery

Vasco failure does not bring down services.



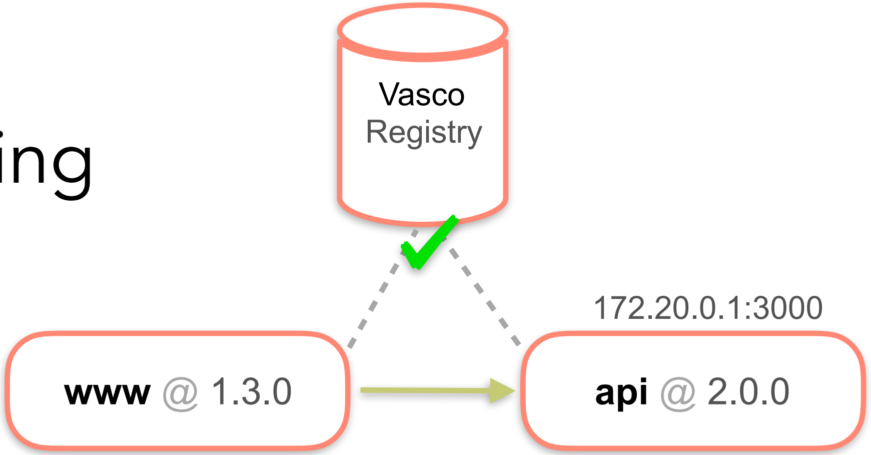
Feature: Client-side Discovery

Vasco failure does not bring down services.



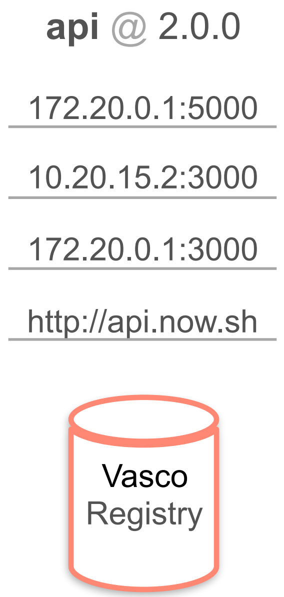
Feature: Client-side Discovery

Vasco failure does not bring down services.



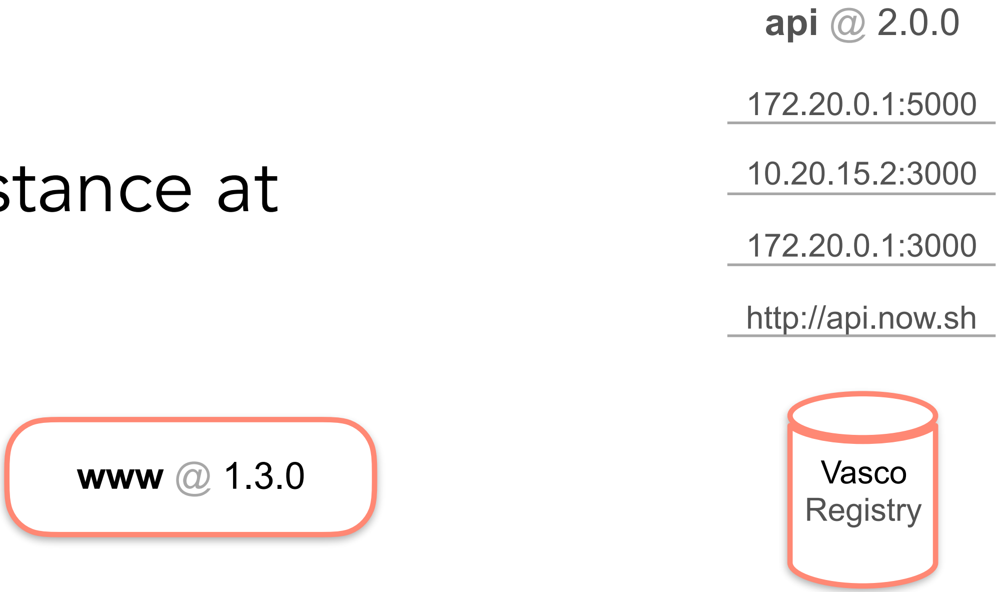
Feature: Load Balancing

Pick a healthy instance at random.



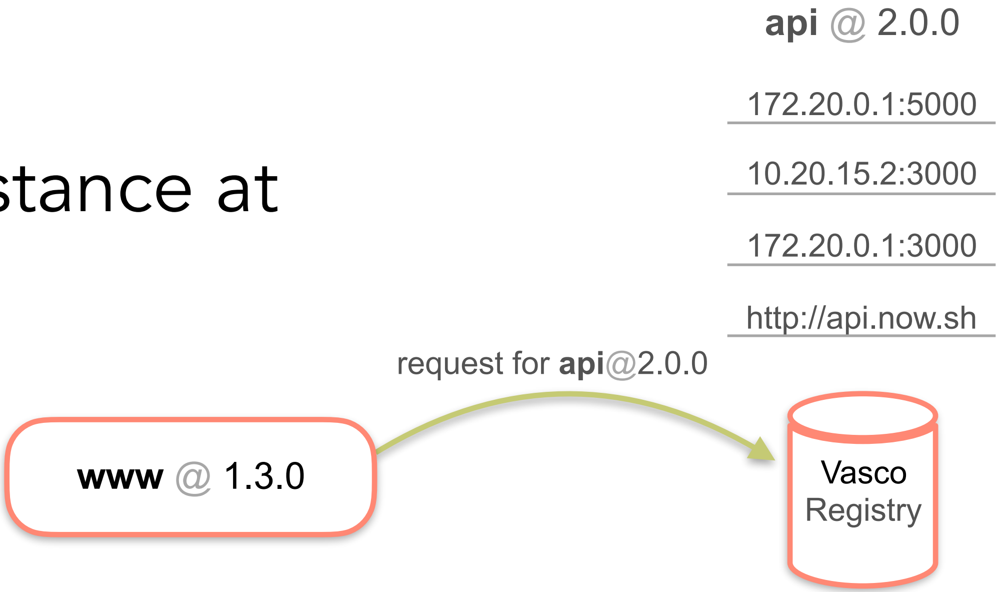
Feature: Load Balancing

Pick a healthy instance at random.



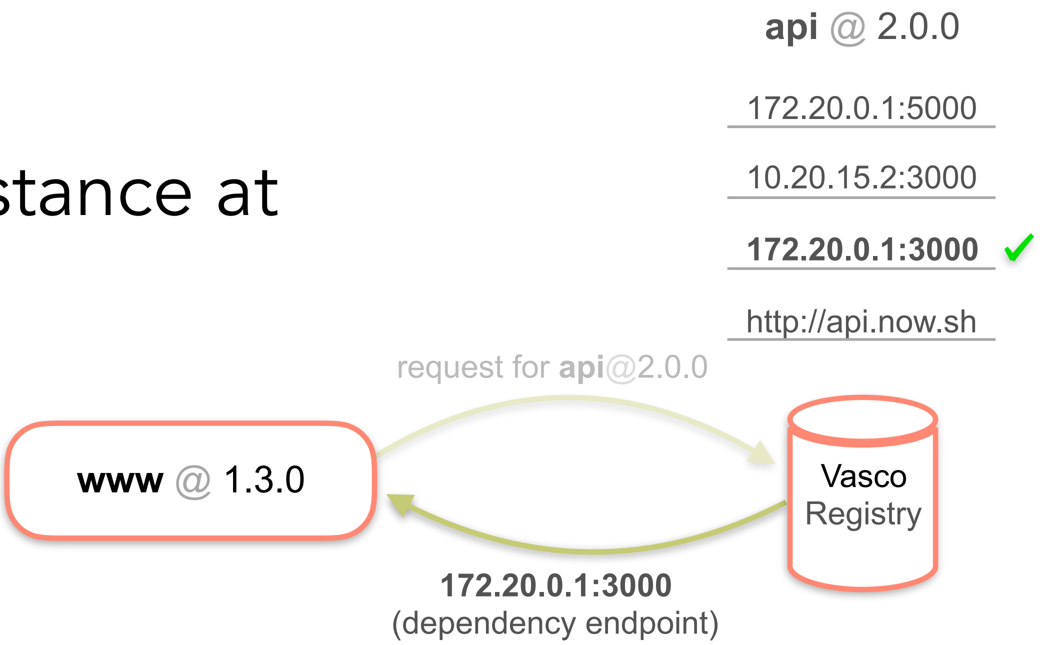
Feature: Load Balancing

Pick a healthy instance at random.



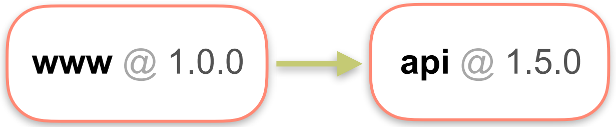
Feature: Load Balancing

Pick a healthy instance at random.



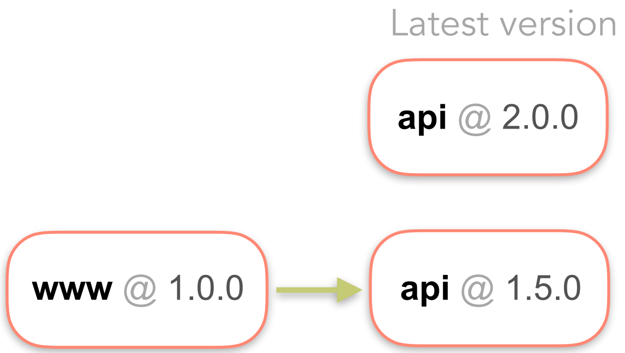
Features: Service Versioning

Multiple versions remain active in production.



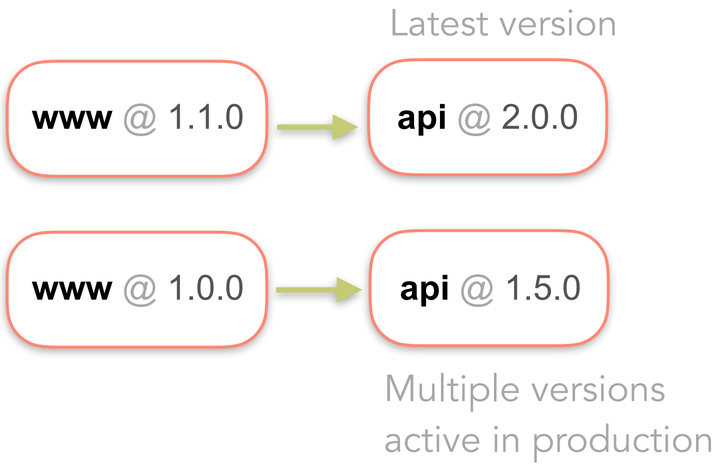
Features: Service Versioning

Multiple versions remain active in production.



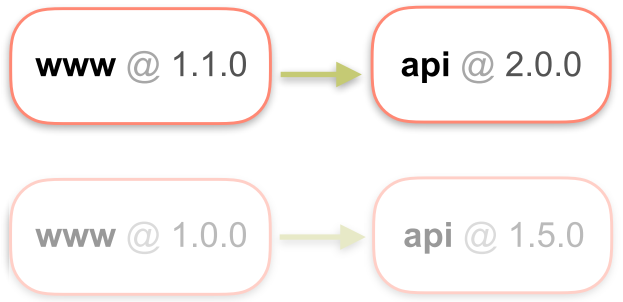
Features: Service Versioning

Multiple versions remain active in production.



Features: Service Versioning

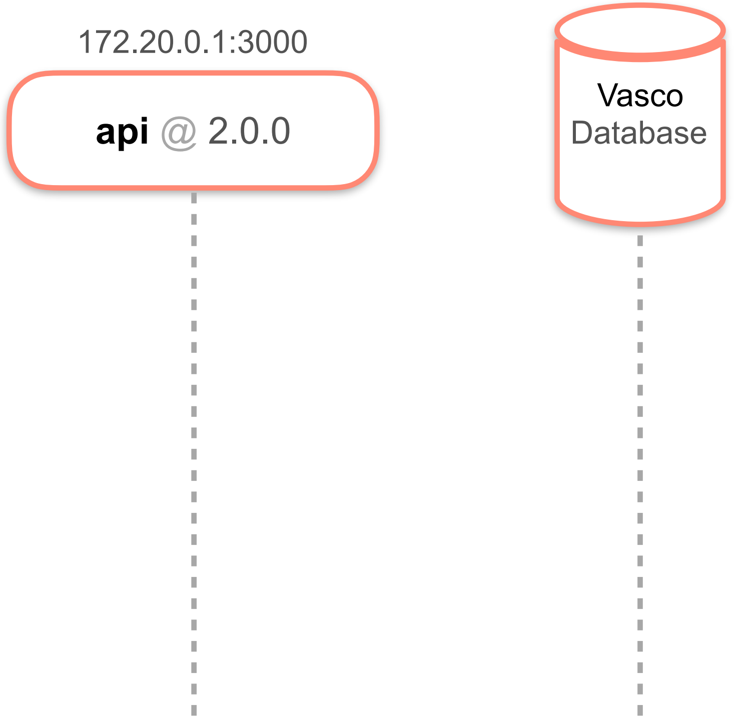
Multiple versions remain active in production.



Old version slowly going out of use, as clients stop using it

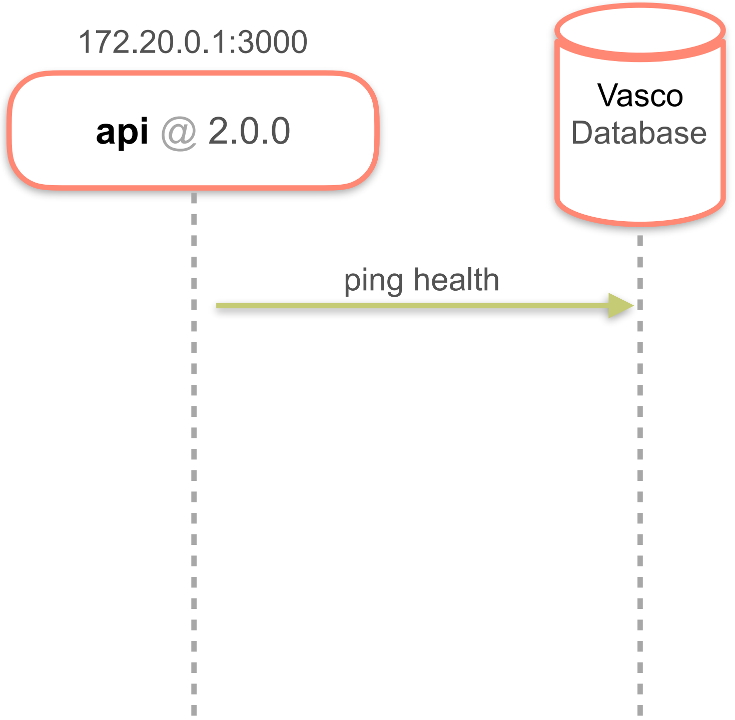
Features: Health Status

Periodically pings Registry with health.



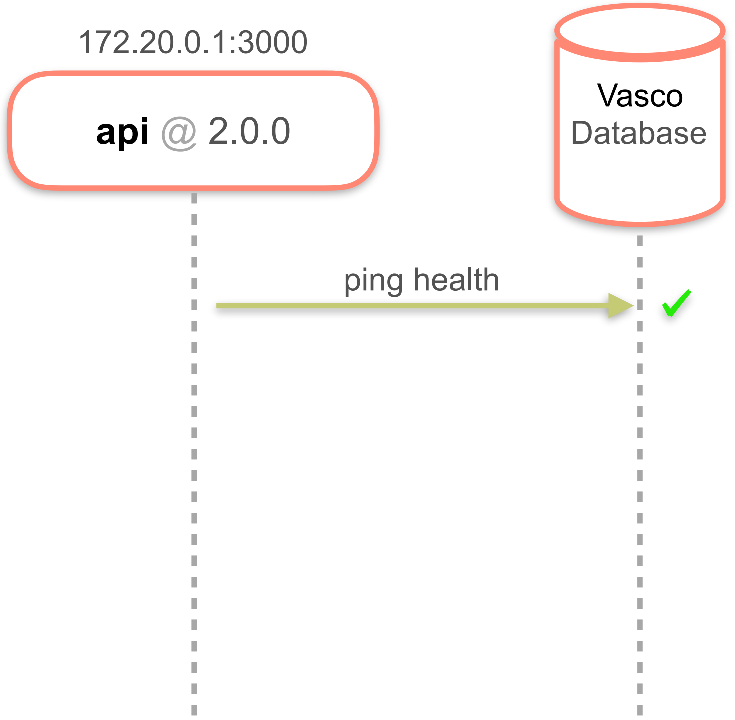
Features: Health Status

Periodically pings Registry with health.



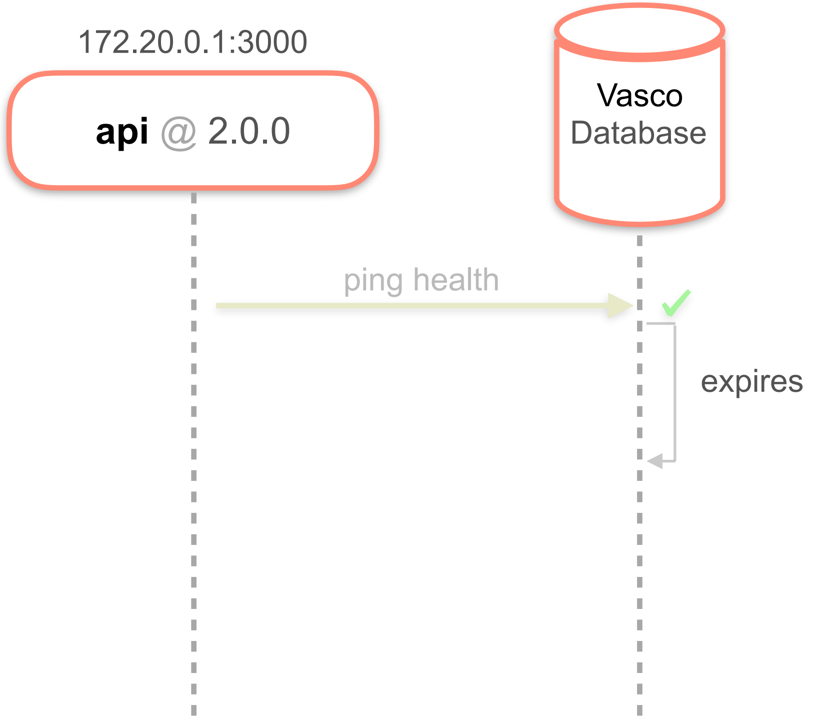
Features: Health Status

Periodically pings Registry with health.



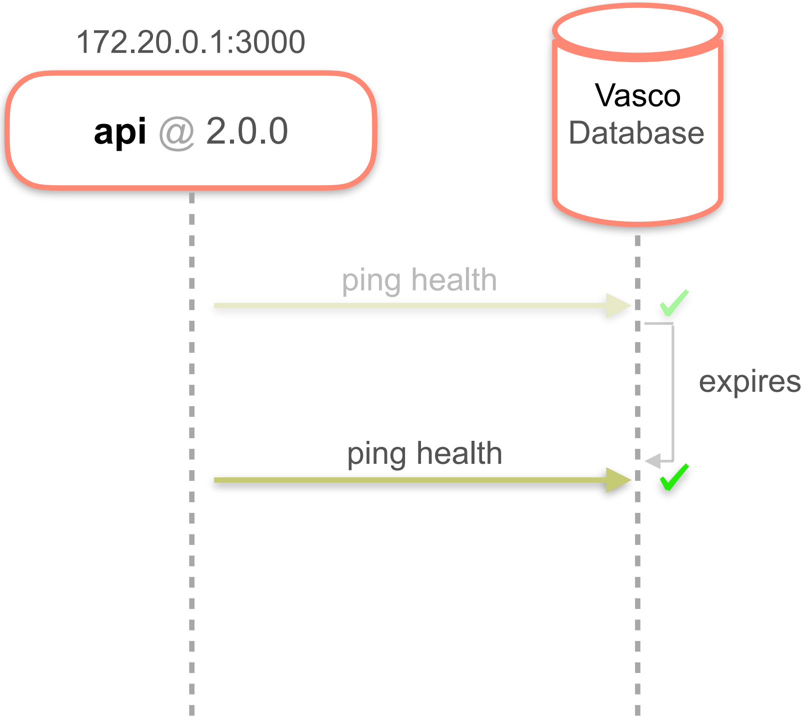
Features: Health Status

Periodically pings Registry with health.



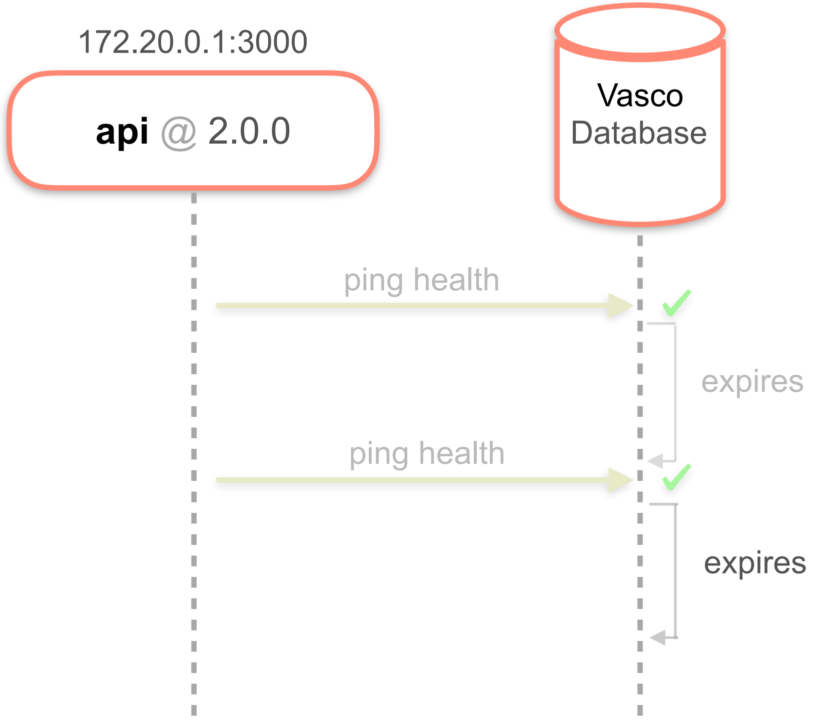
Features: Health Status

Periodically pings Registry with health.



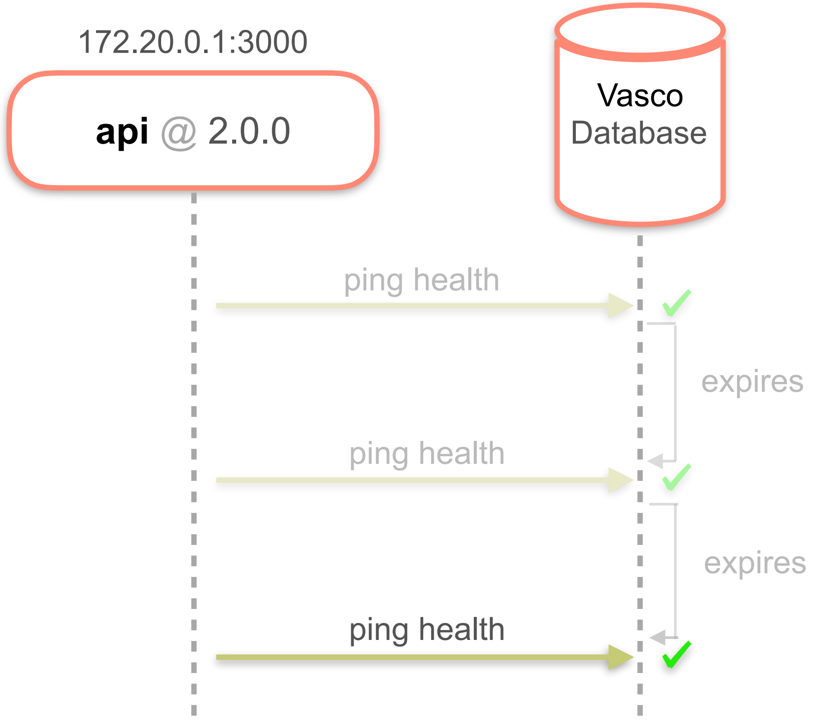
Features: Health Status

Periodically pings Registry with health.



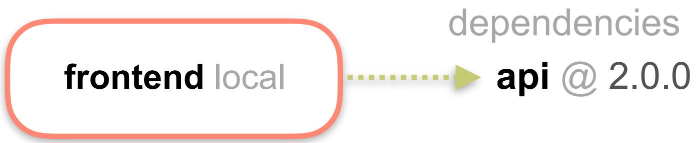
Features: Health Status

Periodically pings Registry with health.



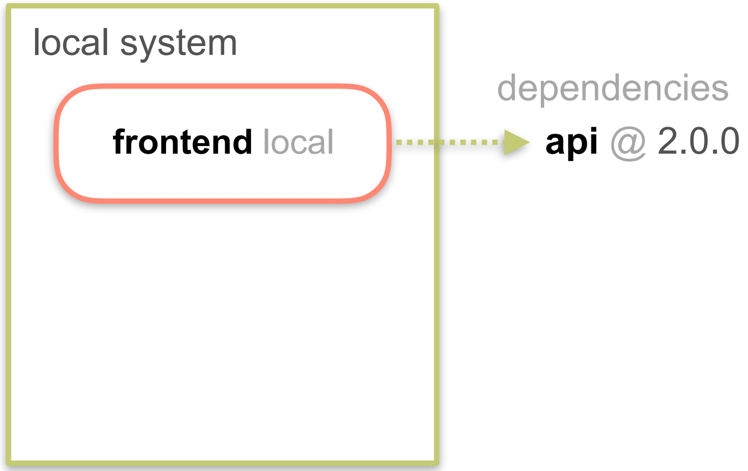
Feature: Easy Mocking

Must be able to develop locally.



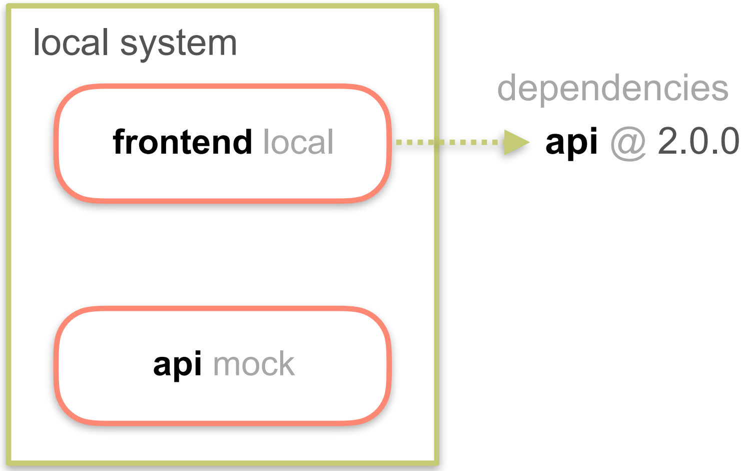
Feature: Easy Mocking

Must be able to develop locally.



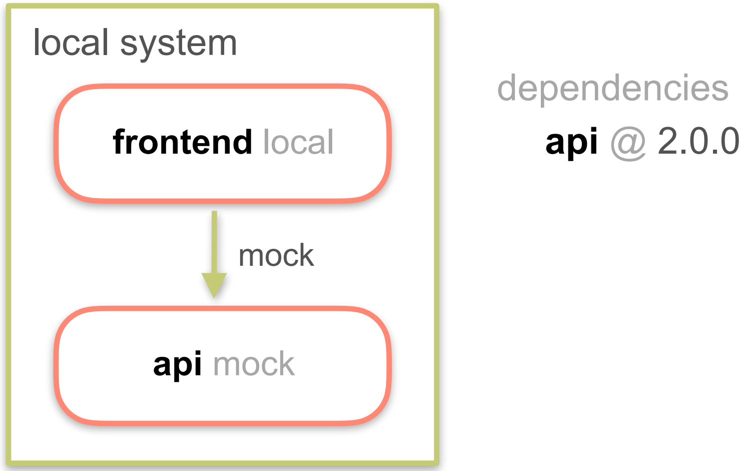
Feature: Easy Mocking

Must be able to develop locally.



Feature: Easy Mocking

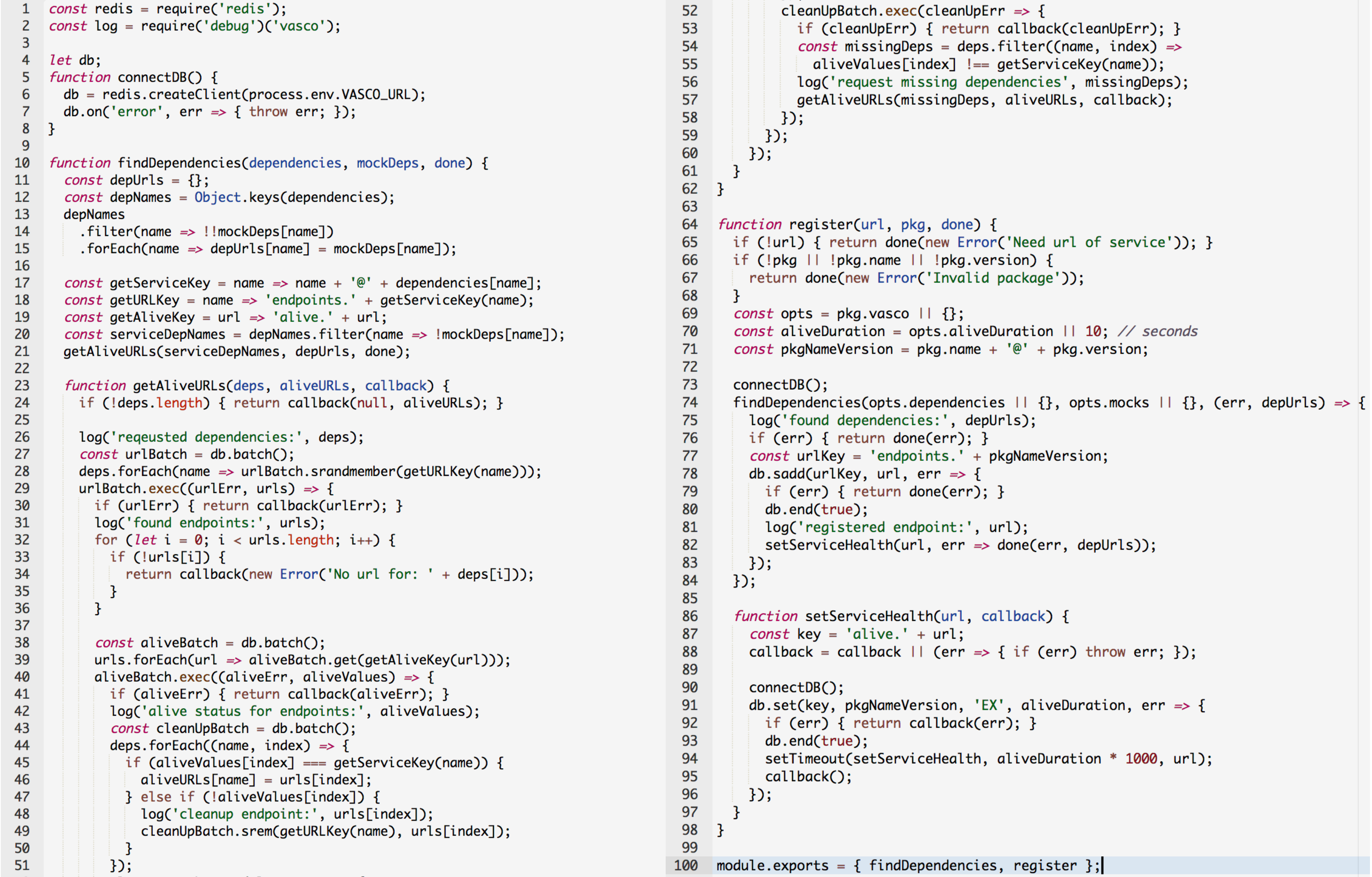
Must be able to develop locally.



Demo Time!

THIS IS ALL THE CODE

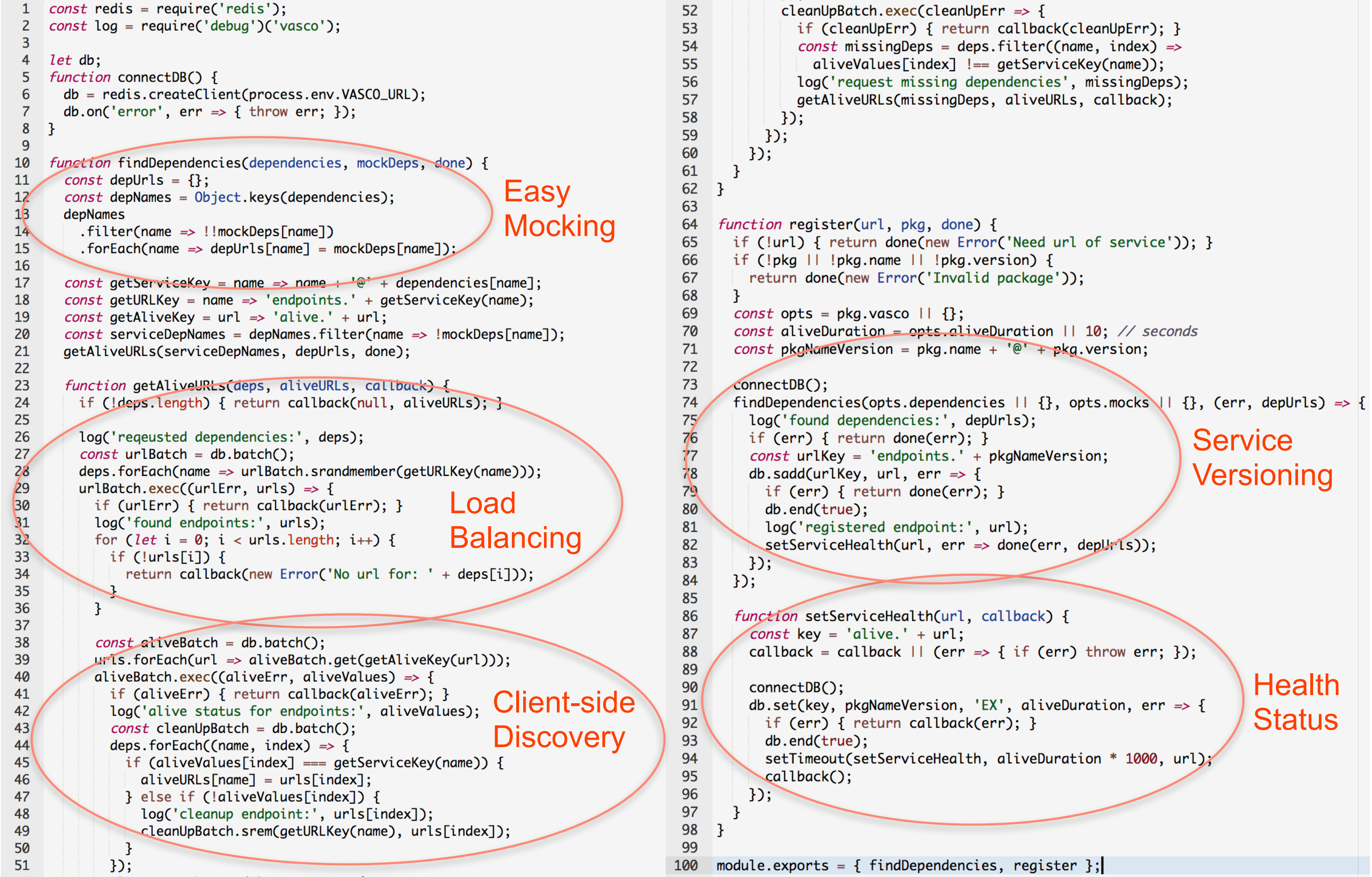
100 LINES OF NODE



```javascript
const redis = require('redis');
const log = require('debug')('vasco');

let db;
function connectDB() {
  db = redis.createClient(process.env.VASCO_URL);
  db.on('error', err => { throw err; });
}

function findDependencies(dependencies, mockDeps, done) {
  const depUrls = {};
  const depNames = Object.keys(dependencies);
  depNames
    .filter(name => !!mockDeps[name])
    .forEach(name => depUrls[name] = mockDeps[name]);

  const getServiceKey = name => name + '@' + dependencies[name];
  const getURLKey = name => 'endpoints.' + getServiceKey(name);
  const getAliveKey = url => 'alive.' + url;
  const serviceDepNames = depNames.filter(name => !mockDeps[name]);
  getAliveURLs(serviceDepNames, depUrls, done);

  function getAliveURLs(deps, aliveURLs, callback) {
    if (!deps.length) { return callback(null, aliveURLs); }

    log('reqeusted dependencies:', deps);
    const urlBatch = db.batch();
    deps.forEach(name => urlBatch.srandmember(getURLKey(name)));
    urlBatch.exec((urlErr, urls) => {
      if (urlErr) { return callback(urlErr); }
      log('found endpoints:', urls);
      for (let i = 0; i < urls.length; i++) {
        if (!urls[i]) {
          return callback(new Error('No url for: ' + deps[i]));
        }
      }

      const aliveBatch = db.batch();
      urls.forEach(url => aliveBatch.get(getAliveKey(url)));
      aliveBatch.exec((aliveErr, aliveValues) => {
        if (aliveErr) { return callback(aliveErr); }
        log('alive status for endpoints:', aliveValues);
        const cleanUpBatch = db.batch();
        deps.forEach((name, index) => {
          if (aliveValues[index] === getServiceKey(name)) {
            aliveURLs[name] = urls[index];
          } else if (!aliveValues[index]) {
            log('cleanup endpoint:', urls[index]);
            cleanUpBatch.srem(getURLKey(name), urls[index]);
          }
        });
        cleanUpBatch.exec(cleanUpErr => {
          if (cleanUpErr) { return callback(cleanUpErr); }
          const missingDeps = deps.filter((name, index) =>
            aliveValues[index] !== getServiceKey(name));
          log('request missing dependencies', missingDeps);
          getAliveURLs(missingDeps, aliveURLs, callback);
        });
      });
    });
  }
}

function register(url, pkg, done) {
  if (!url) { return done(new Error('Need url of service')); }
  if (!pkg || !pkg.name || !pkg.version) {
    return done(new Error('Invalid package'));
  }
  const opts = pkg.vasco || {};
  const aliveDuration = opts.aliveDuration || 10; // seconds
  const pkgNameVersion = pkg.name + '@' + pkg.version;

  connectDB();
  findDependencies(opts.dependencies || {}, opts.mocks || {}, (err, depUrls) => {
    log('found dependencies:', depUrls);
    if (err) { return done(err); }
    const urlKey = 'endpoints.' + pkgNameVersion;
    db.sadd(urlKey, url, err => {
      if (err) { return done(err); }
      db.end(true);
      log('registered endpoint:', url);
      setServiceHealth(url, err => done(err, depUrls));
    });
  });

  function setServiceHealth(url, callback) {
    const key = 'alive.' + url;
    callback = callback || (err => { if (err) throw err; });

    connectDB();
    db.set(key, pkgNameVersion, 'EX', aliveDuration, err => {
      if (err) { return callback(err); }
      db.end(true);
      setTimeout(setServiceHealth, aliveDuration * 1000, url);
      callback();
    });
  }
}

module.exports = { findDependencies, register };
```

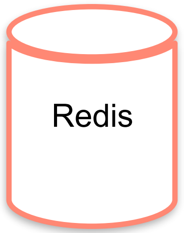
THIS IS ALL THE CODE

100 LINES OF NODE

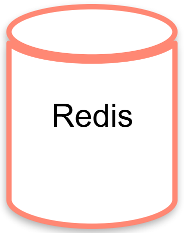


```javascript
const redis = require('redis');
const log = require('debug')('vasco');

let db;
function connectDB() {
  db = redis.createClient(process.env.VASCO_URL);
  db.on('error', err => { throw err; });
}

function findDependencies(dependencies, mockDeps, done) {
  const depUrls = {};
  const depNames = Object.keys(dependencies);
  depNames
    .filter(name => !!mockDeps[name])
    .forEach(name => depUrls[name] = mockDeps[name]);

  const getServiceKey = name => name + '@' + dependencies[name];
  const getURLKey = name => 'endpoints.' + getServiceKey(name);
  const getAliveKey = url => 'alive.' + url;
  const serviceDepNames = depNames.filter(name => !mockDeps[name]);
  getAliveURLs(serviceDepNames, depUrls, done);

  function getAliveURLs(deps, aliveURLs, callback) {
    if (!deps.length) { return callback(null, aliveURLs); }

    log('reqeusted dependencies:', deps);
    const urlBatch = db.batch();
    deps.forEach(name => urlBatch.srandmember(getURLKey(name)));
    urlBatch.exec((urlErr, urls) => {
      if (urlErr) { return callback(urlErr); }
      log('found endpoints:', urls);
      for (let i = 0; i < urls.length; i++) {
        if (!urls[i]) {
          return callback(new Error('No url for: ' + deps[i]));
        }
      }

      const aliveBatch = db.batch();
      urls.forEach(url => aliveBatch.get(getAliveKey(url)));
      aliveBatch.exec((aliveErr, aliveValues) => {
        if (aliveErr) { return callback(aliveErr); }
        log('alive status for endpoints:', aliveValues);
        const cleanUpBatch = db.batch();
        deps.forEach((name, index) => {
          if (aliveValues[index] === getServiceKey(name)) {
            aliveURLs[name] = urls[index];
          } else if (!aliveValues[index]) {
            log('cleanup endpoint:', urls[index]);
            cleanUpBatch.srem(getURLKey(name), urls[index]);
          }
        });
        cleanUpBatch.exec(cleanUpErr => {
          if (cleanUpErr) { return callback(cleanUpErr); }
          const missingDeps = deps.filter((name, index) =>
            aliveValues[index] !== getServiceKey(name));
          log('request missing dependencies', missingDeps);
          getAliveURLs(missingDeps, aliveURLs, callback);
        });
      });
    });
  }
}

function register(url, pkg, done) {
  if (!url) { return done(new Error('Need url of service')); }
  if (!pkg || !pkg.name || !pkg.version) {
    return done(new Error('Invalid package'));
  }
  const opts = pkg.vasco || {};
  const aliveDuration = opts.aliveDuration || 10; // seconds
  const pkgNameVersion = pkg.name + '@' + pkg.version;

  connectDB();
  findDependencies(opts.dependencies || {}, opts.mocks || {}, (err, depUrls) => {
    log('found dependencies:', depUrls);
    if (err) { return done(err); }
    const urlKey = 'endpoints.' + pkgNameVersion;
    db.sadd(urlKey, url, err => {
      if (err) { return done(err); }
      db.end(true);
      log('registered endpoint:', url);
      setServiceHealth(url, err => done(err, depUrls));
    });
  });

  function setServiceHealth(url, callback) {
    const key = 'alive.' + url;
    callback = callback || (err => { if (err) throw err; });

    connectDB();
    db.set(key, pkgNameVersion, 'EX', aliveDuration, err => {
      if (err) { return callback(err); }
      db.end(true);
      setTimeout(setServiceHealth, aliveDuration * 1000, url);
      callback();
    });
  }
}

module.exports = { findDependencies, register };
```

Architecture: Vasco Registry

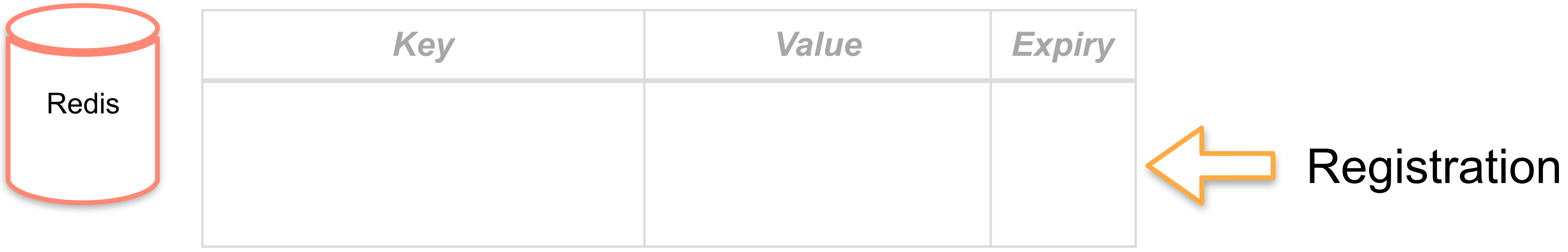


Architecture: Vasco Registry

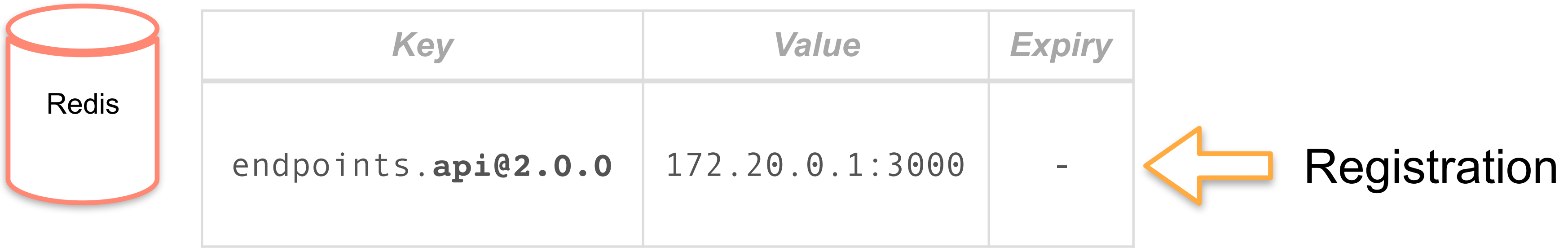


Key	*Value*	*Expiry*

Architecture: Vasco Registry

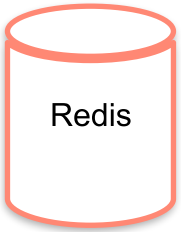


Architecture: Vasco Registry



Key	*Value*	*Expiry*
`endpoints.`**`api@2.0.0`**	`172.20.0.1:3000`	-

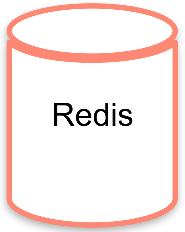
Architecture: Vasco Registry



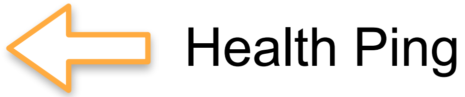
Key	*Value*	*Expiry*
endpoints.**api@2.0.0**	172.20.0.1:3000	-

Health Ping

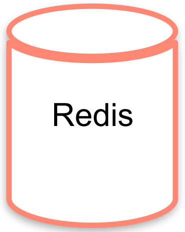
Architecture: Vasco Registry



Key	*Value*	*Expiry*
`endpoints.`**`api@2.0.0`**	`172.20.0.1:3000`	-
`alive.`**`172.20.0.1:3000`**	`api@2.0.0`	10



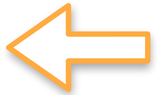
Architecture: Vasco Registry



Key	*Value*	*Expiry*

`endpoints.`**`api@2.0.0`**	`172.20.0.1:3000`
	`10.20.50.2:5000`
		-

`alive.`**`172.20.0.1:3000`**	`api@2.0.0`	10



Registration

Architecture: Vasco Registry

Redis

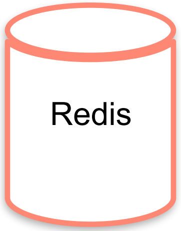
Key	*Value*	*Expiry*

`endpoints.`**`api@2.0.0`**	`172.20.0.1:3000`
	`10.20.50.2:5000`
		-

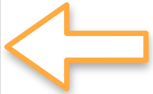
`alive.`**`172.20.0.1:3000`**	`api@2.0.0`	10
`alive.`**`10.20.50.2:5000`**	`api@2.0.0`	10

Health Ping

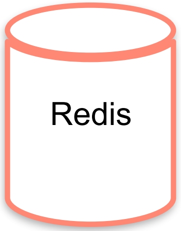
Architecture: Vasco Registry



Key	*Value*	*Expiry*
endpoints.**api@2.0.0**	172.20.0.1:3000 10.20.50.2:5000	-
~~alive.**172.20.0.1:3000**~~	~~api@2.0.0~~	~~10~~
alive.**10.20.50.2:5000**	api@2.0.0	10

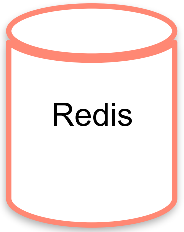


Architecture: Vasco Registry



Key	*Value*	*Expiry*
endpoints.**api@2.0.0**	172.20.0.1:3000 10.20.50.2:5000	-
alive.**10.20.50.2:5000**	api@2.0.0	10

Architecture: Vasco Registry



Key	*Value*	*Expiry*
endpoints.**api@2.0.0**	172.20.0.1:3000 10.20.50.2:5000	-
alive.**10.20.50.2:5000**	api@2.0.0	10

Configuration: Startup Script

```
$ VASCO_URL=redis://... node server.js
```

Configuration: Startup Script

```
$ VASCO_URL=redis://... node server.js
```

Redis Connection String

Configuration: Startup Script

```
$ VASCO_URL=redis://... node server.js
```

Redis Connection String

Amazon Elasticache URL

Configuration: Metadata

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0"
}
```

Configuration: Metadata

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {}
}
```

Configuration: Dependencies

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {
    "dependencies": {
      "api": "2.0.0",
      "auth": "1.0.0"
    }
  }
}
```

Configuration: Mocks

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {
    "dependencies": {
      "api": "2.0.0",
      "auth": "1.0.0"
    },
    "mocks": {
      "auth": "localhost:3000"
    }
  }
}
```

Configuration: Mocks

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {
    "dependencies": {
      "api": "2.0.0",
      "auth": "1.0.0"
    },
    "mocks": {
      "auth": "localhost:3000"
    }
  }
}
```

Configuration: Health Expiry

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {
    "dependencies": {
      "api": "2.0.0",
      "auth": "1.0.0"
    },
    "mocks": {
      "auth": "localhost:3000"
    },
    "aliveDuration": 10
  }
}
```

Configuration: Health Expiry

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {
    "dependencies": {
      "api": "2.0.0",
      "auth": "1.0.0"
    },
    "mocks": {
      "auth": "localhost:3000"
    },
    "aliveDuration": 10
  }
}
```

Configuration: All together

```
// package.json
{
  "name": "frontend",
  "version": "1.3.0",

  "vasco": {
    "dependencies": {
      "api": "2.0.0",
      "auth": "1.0.0"
    },
    "mocks": {
      "auth": "localhost:3000"
    },
    "aliveDuration": 10
  }
}
```

Architecture: API Surface

```
module.exports = {
  register,
  findDependencies
};
```

Architecture: API Surface

```
module.exports = {
  register,
  findDependencies
};
```

Registration

Architecture: API Surface

```
module.exports = {
  register,
  findDependencies
};
```

Registration

Discovery

Architecture: API Surface

```
function register(
    url,            // service url to register
    pkg,            // name, version, vasco config
    done            // asynchronous callback
 ) {}
```

```
module.exports = {
  register,
  findDependencies
};
```

Architecture: API Surface

```
module.exports = {
  register,
  findDependencies
};
```

```
function register(
    url,                // service url to register
    pkg,                // name, version, vasco config
    done                // asynchronous callback
 ) {}
```

```
function findDependencies(
    dependencies,   // service dependencies
    mockDeps,       // optional mocks
    done            // asynchronous callback
 ) {}
```

Implementation: Service Versioning

```
function register(url, { name, version, vasco }, done) {

  const pkgNameVersion = pkg.name + '@' + pkg.version;
  const urlKey = 'endpoints.' + pkgNameVersion;

  // find service dependency urls
  findDependencies(vasco.dependencies, vasco.mocks, (depUrls) => ...

    // add service to registry
    db.sadd(urlKey, url, ...

      done(null, depUrls)
}
```

Implementation: Service Versioning

```
function register(url, { name, version, vasco }, done) {

  const pkgNameVersion = pkg.name + '@' + pkg.version;
  const urlKey = 'endpoints.' + pkgNameVersion;

  // find service dependency urls
  findDependencies(vasco.dependencies, vasco.mocks, (depUrls) => ...

    // add service to registry
    db.sadd(urlKey, url, ...

      done(null, depUrls)
}
```

Implementation: Service Versioning

```
function register(url, { name, version, vasco }, done) {

  const pkgNameVersion = pkg.name + '@' + pkg.version;
  const urlKey = 'endpoints.' + pkgNameVersion;

  // find service dependency urls
  findDependencies(vasco.dependencies, vasco.mocks, (depUrls) => ...

    // add service to registry
    db.sadd(urlKey, url, ...

      done(null, depUrls)
}
```

Implementation: Service Versioning

```
function register(url, { name, version, vasco }, done) {

  const pkgNameVersion = pkg.name + '@' + pkg.version;
  const urlKey = 'endpoints.' + pkgNameVersion;

  // find service dependency urls
  findDependencies(vasco.dependencies, vasco.mocks, (depUrls) => ...

    // add service to registry
    db.sadd(urlKey, url, ...

      done(null, depUrls)
}
```

Implementation: Service Versioning

```
function register(url, { name, version, vasco }, done) {

  const pkgNameVersion = pkg.name + '@' + pkg.version;
  const urlKey = 'endpoints.' + pkgNameVersion;

  // find service dependency urls
  findDependencies(vasco.dependencies, vasco.mocks, (depUrls) => ...

    // add service to registry
    db.sadd(urlKey, url, ...

      done(null, depUrls)
}
```

Implementation: Health Status

```
function setServiceHealth(url, callback) {

  // connect to db
  connectDB();

  // set key alive.<instance> with expiry
  db.set(aliveKey, pkgNameVersion, 'EX', aliveDuration, ...

    // close db connection
    db.end(true);

    // recursive call, automatically slows down under load
    setTimeout(setServiceHealth, aliveDuration * 1000, url)
}
```

Implementation: Health Status

```
function setServiceHealth(url, callback) {

  // connect to db
  connectDB();

  // set key alive.<instance> with expiry
  db.set(aliveKey, pkgNameVersion, 'EX', aliveDuration, ...

    // close db connection
    db.end(true);

    // recursive call, automatically slows down under load
    setTimeout(setServiceHealth, aliveDuration * 1000, url)
}
```

Implementation: Health Status

```
function setServiceHealth(url, callback) {

  // connect to db
  connectDB();

  // set key alive.<instance> with expiry
  db.set(aliveKey, pkgNameVersion, 'EX', aliveDuration, ...

    // close db connection
    db.end(true);

    // recursive call, automatically slows down under load
    setTimeout(setServiceHealth, aliveDuration * 1000, url)
}
```

Implementation: Health Status

```
function setServiceHealth(url, callback) {

  // connect to db
  connectDB();

  // set key (alive.<instance>) to <service> with expiry
  db.set(aliveKey, pkgNameVersion, 'EX', aliveDuration, ...

    // close db connection
    db.end(true);

    // recursive call, automatically slows down under load
    setTimeout(setServiceHealth, aliveDuration * 1000, url)
}
```

Implementation: Easy Mocking

```
function findDependencies(dependencies, mockDeps, done) {

    // dependency URL map (final output)
    const depUrls = {};

    // list of requested dependencies
    const depNames = Object.keys(dependencies);

    // set mocked dependencies already
    depNames
      .filter(name => !!mockDeps[name])
      .forEach(name => depUrls[name] = mockDeps[name]);

    ...
}
```

Implementation: Easy Mocking

```
function findDependencies(dependencies, mockDeps, done) {

    // dependency URL map (final output)
    const depUrls = {};

    // list of requested dependencies
    const depNames = Object.keys(dependencies);

    // set mocked dependencies already
    depNames
      .filter(name => !!mockDeps[name])
      .forEach(name => depUrls[name] = mockDeps[name]);

    ...
}
```

Implementation: Easy Mocking

```
function findDependencies(dependencies, mockDeps, done) {

    // dependency URL map (final output)
    const depUrls = {};

    // list of requested dependencies
    const depNames = Object.keys(dependencies);

    // set mocked dependencies already
    depNames
      .filter(name => !!mockDeps[name])
      .forEach(name => depUrls[name] = mockDeps[name]);

    ...
}
```

Implementation: Easy Mocking

```
function findDependencies(dependencies, mockDeps, done) {

    // dependency URL map (final output)
    const depUrls = {};

    // list of requested dependencies
    const depNames = Object.keys(dependencies);

    // set mocked dependencies already
    depNames
      .filter(name => !!mockDeps[name])
      .forEach(name => depUrls[name] = mockDeps[name]);

    ...
}
```

Implementation: Easy Mocking

```
function findDependencies(dependencies, mockDeps, done) {

    // dependency URL map (final output)
    const depUrls = {};

    // list of requested dependencies
    const depNames = Object.keys(dependencies);

    // set mocked dependencies already
    depNames
      .filter(name => !!mockDeps[name])
      .forEach(name => depUrls[name] = mockDeps[name]);

    ...
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

Implementation: Client-side Discovery

```
function getAliveURLs(deps, aliveURLs, callback) {

  // recursion base case, return if no requested dependencies
  if (!deps.length) { return callback(null, aliveURLs); }

  // get a random service instance for each dependency
  deps.forEach(name => urlBatch.srandmember(getURLKey(name)));

    // check whether instances are actually alive
    urls.forEach(url => aliveBatch.get(getAliveKey(url)));

      // remove those not alive, or mapped to wrong <service>
      cleanUpBatch.srem(getURLKey(name), urls[index]);

        // find new urls for removed dependencies, recursively
        getAliveURLs(missingDeps, aliveURLs, callback);
}
```

You can make stuff!

Anchors

- vasco
  https://github.com/asyncanup/vasco

- vasco-frontend
  https://github.com/asyncanup/vasco-frontend

- Some History and Patterns in Service Discovery
  https://www.youtube.com/watch?v=PptS7EgQvx4

That's all Folks!