



Node.js Microservices on Autopilot

Wyatt Preul – jsgeek.com/jsconf

One day Chicken-micro was making a request to the corn server when—boom! The request timed out.



Goodness gracious, the servers
must be down. I must go and tell
the System Operator.

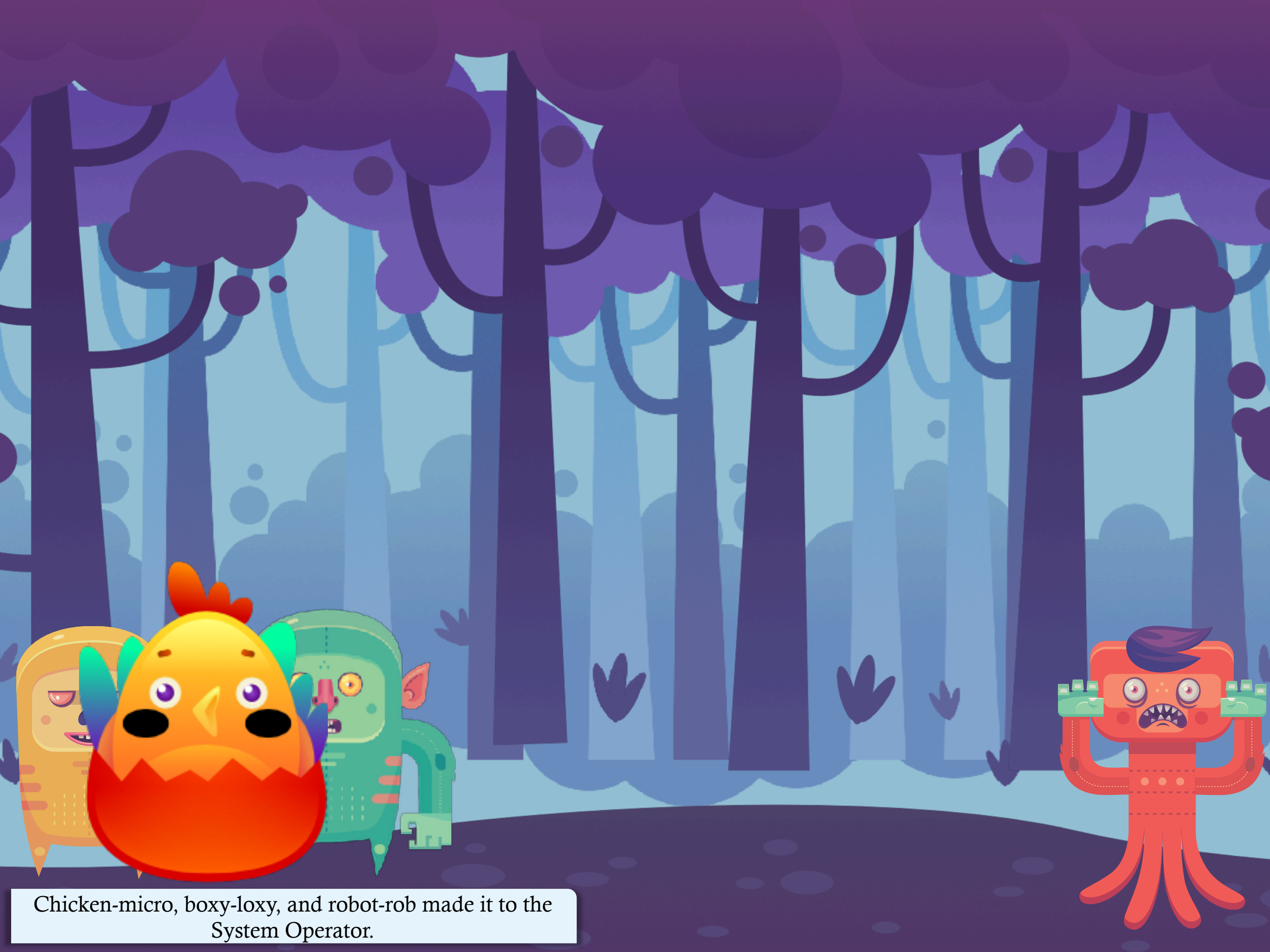




Chicken-micro went along until she came to boxy-loxy.



Chicken-micro and boxy-loxy went along until they came to robot-rob



Chicken-micro, boxy-loxy, and robot-rob made it to the
System Operator.

Chicken-micro Postmortem

- Overloaded server should have returned 503
- Load-balancer should have taken slow server out of rotation
- How would chicken-micro know that there are other healthy servers and that the next request will succeed?

Do microservices help?

- Hypothetical Steps
 - Decompose the corn service into many smaller services
 - Each microservice is independently deployable
 - New microservices are each located behind load balancers
- When microservices are architected the same as the services they replace, they will face the same challenges as the previous services.

What is a Microservice?

- Small, decomposed, isolated and independently deployable services
- Stateless and less fragile when changes are introduced

Benefits of Microservices

- Embrace failure, works in spite of external failures
- Iterate quickly - disposable services, independently deployable services

Microservice AntiPatterns

- Load balancer between microservices
- Startup order matters
- Load balancers everywhere
- + many others, Microservices AntiPatterns and Pitfalls by Mark Richards

Load balancer between microservices

```
const getCorn = function (id, cb) {  
  const url = `http://corn.prod.site.com/${id}`;  
  wreck.get(url, { timeout: 5000 }, (err, res, corn) => {  
    if (err && err.message === 'Client request timeout') {  
      // retry the request  
      return setTimeout(() => getCorn(id, cb), 1000);  
    }  
    // ... handle err or success  
  });  
};
```



Load balancer between microservices

```
const getCorn = function (id, cb) {  
  const url = `http://corn.prod.site.com/${id}`;  
  wreck.get(url, (err, res, corn) => {  
    // server is under load, retry  
    if (err && err.output.statusCode === 503) {  
      return setTimeout(() => getCorn(id, cb), 1000);  
    }  
    // ... handle err or success  
  });  
};
```



Startup order matters

```
const mysql = require('mysql');

const connection = mysql.createConnection({
  host: process.env.DB_HOST
});

connection.connect();
// no fallback for db not found
```



Load balancers everywhere

```
const config = {  
  prod: {  
    motion: 'motion.prod.srv.site.com',  
    humidity: 'humidity.prod.srv.site.com',  
    temperature: 'temperature.prod.srv.site.com',  
    frontend: 'frontend.prod.srv.site.com',  
    logs: 'logs.prod.srv.site.com'  
  }  
}
```





Autopilot Pattern

- Apps that can be deployed and scaled with a single click.
- Apps and workflows that work the same on our laptops as in the cloud (public and private cloud).
- Apps and workflows that aren't married to any specific infrastructure or scheduler.
- Further reading at autopilotpattern.io

Autopilot Applications

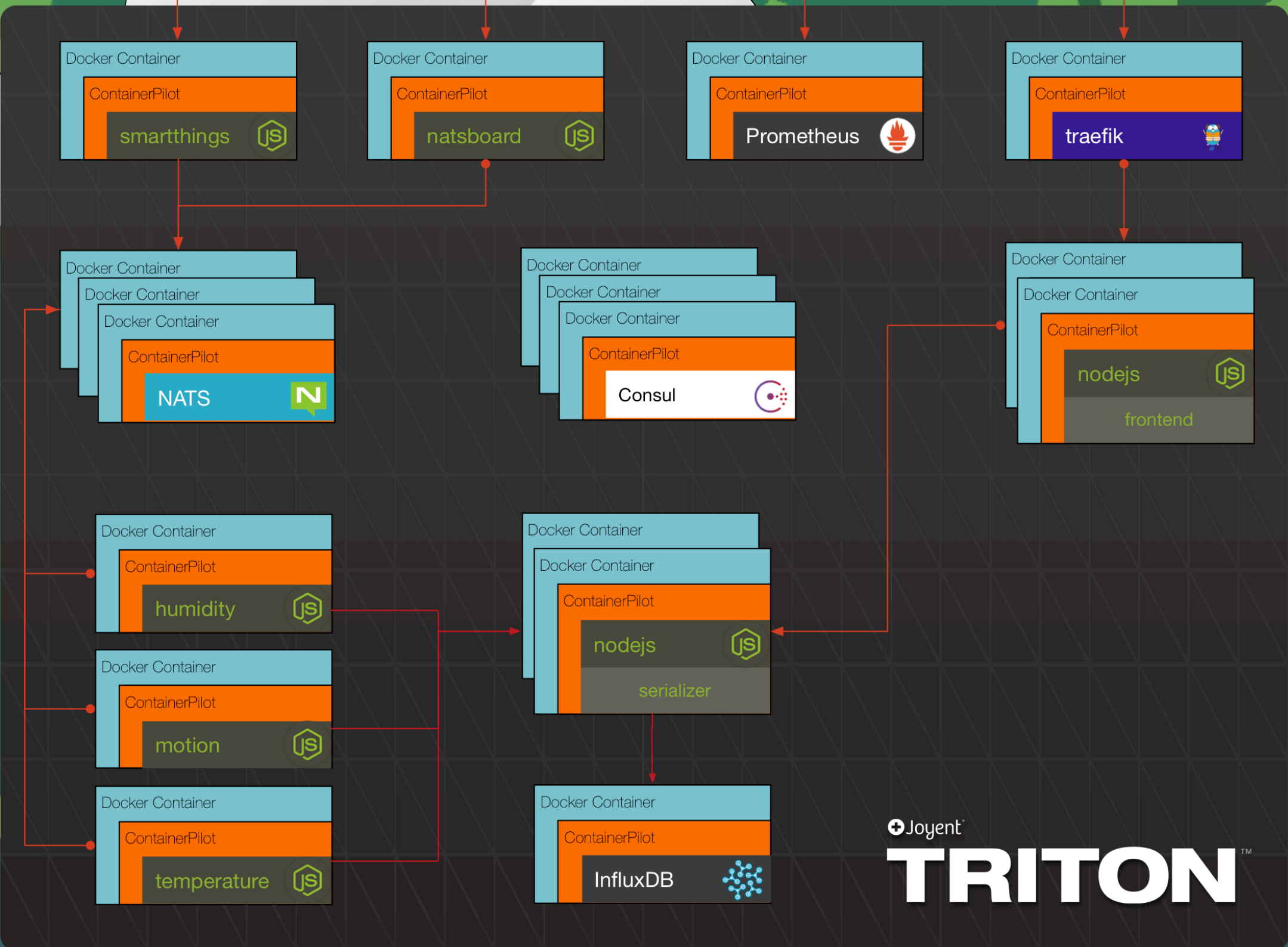
- github.com/autopilotpattern - solutions that follow the Autopilot Pattern
- MongoDB, MySQL, InfluxDB, Consul, NATS, Wordpress, Jenkins, ...
- Container support through ContainerPilot

Autopilot in Practice

- Applications composed of portable docker containers
- Service discovery through consul or another catalog
- Container local health and services respond to service dependency changes

Node.js Example

- <https://github.com/autopilotpattern/nodejs-example>
- Modules used:
 - hapi
 - Seneca
 - Piloted
 - Wreck





Isolation

```
function setupDb () {  
  const influxServer = Piloted.service('influxdb');  
  if (!influxServer) {  
    internals.db = bufferedDb;  
    return;  
  }  
  
  internals.db = new Influx.InfluxDB({  
    host: influxServer.address,  
    port: influxServer.port  
  });  
}
```

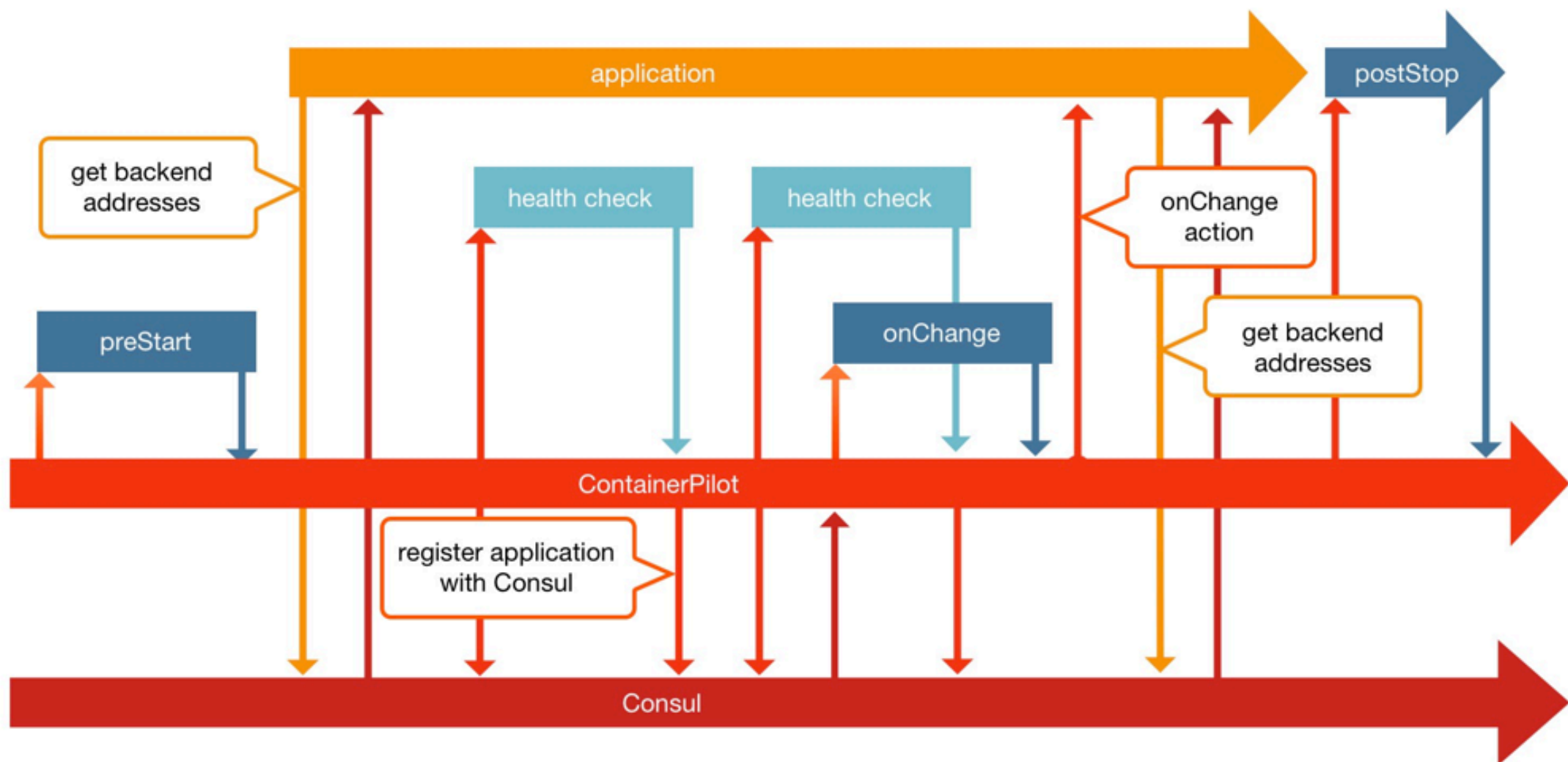
Change

```
Piloted.on('refresh', () => {  
  setupDb();  
});
```

ContainerPilot

- Automates a container's service discovery, life cycle management, and telemetry reporting
- Capabilities
 - Container-local health checks
 - PID 1 init process
 - Service discovery registration and watchers
 - Telemetry reporting to Prometheus
 - Free & Open Source! github.com/joyent/containerpilot

ContainerPilot Lifecycle



ContainerPilot Watches

```
watches: [  
  {  
    name: 'influxdb',  
    interval: 3  
  }  
],  
jobs: [  
  {  
    name: 'onchange-influxdb',  
    exec: 'pkill -SIGHUP node',  
    when: {  
      source: 'watch.influxdb',  
      each: 'changed'  
    }  
  }  
]
```

ContainerPilot Services

```
{  
  name: 'serializer',  
  port: {{.PORT}},  
  exec: 'node /opt/app/',  
  health: {  
    exec: `/usr/bin/curl -o /dev/null --fail  
      -s http://localhost:{{.PORT}}/health`,  
    interval: 2,  
    ttl: 5  
  }  
}
```

Circuit Breakers

- Prevent requests that will fail from taking place and overburdening a service
- Once a response timeout threshold has been reached block future requests to the service until the service is able to catch-up/recover
- Is this possible to implement with a load balancer?

hapi Circuit Breakers

```
const server = new Hapi.Server({
  load: {
    sampleInterval: 50      // milliseconds
  }
});

server.connection({
  port: process.env.PORT,
  load: {
    maxEventLoopDelay: 20   // milliseconds
  }
});
```

Balancing in the Client

```
// Round-robin  
const service = Piloted.service('nats');  
  
// You decide  
const services = Piloted.serviceHosts('nats');  
  
// Grab a random healthy service  
services[Math.floor(Math.random() * services.length)];
```



Load Balancers at the Edge

- Don't expose microservices directly outside of your organization
- Setup a load balancer that can use Consul, in our example this is traefik, could be HA Proxy, nginx, or anything else
- API gateways have a place when exposing the business value we created with our microservices

Telemetry Support

```
telemetry: {  
  port: 9090,  
  tags: ['op'],  
  metrics: [  
    {  
      namespace: 'example',  
      subsystem: 'process',  
      name: 'event_delay',  
      help: 'Node.js event loop delay',  
      type: 'gauge'  
    }  
  ]  
}
```

hapi Metrics with Toppsy



Further Reading:

- All links can be found at: jsgeek.com/jsconf
- Autopilot Pattern - autopilotpattern.io
- hapi - hapijs.com
- ContainerPilot – joyent.com
- Seneca – senecajs.org