Microservices vs Hadoop ecosystem

Marton Elek

2017 february

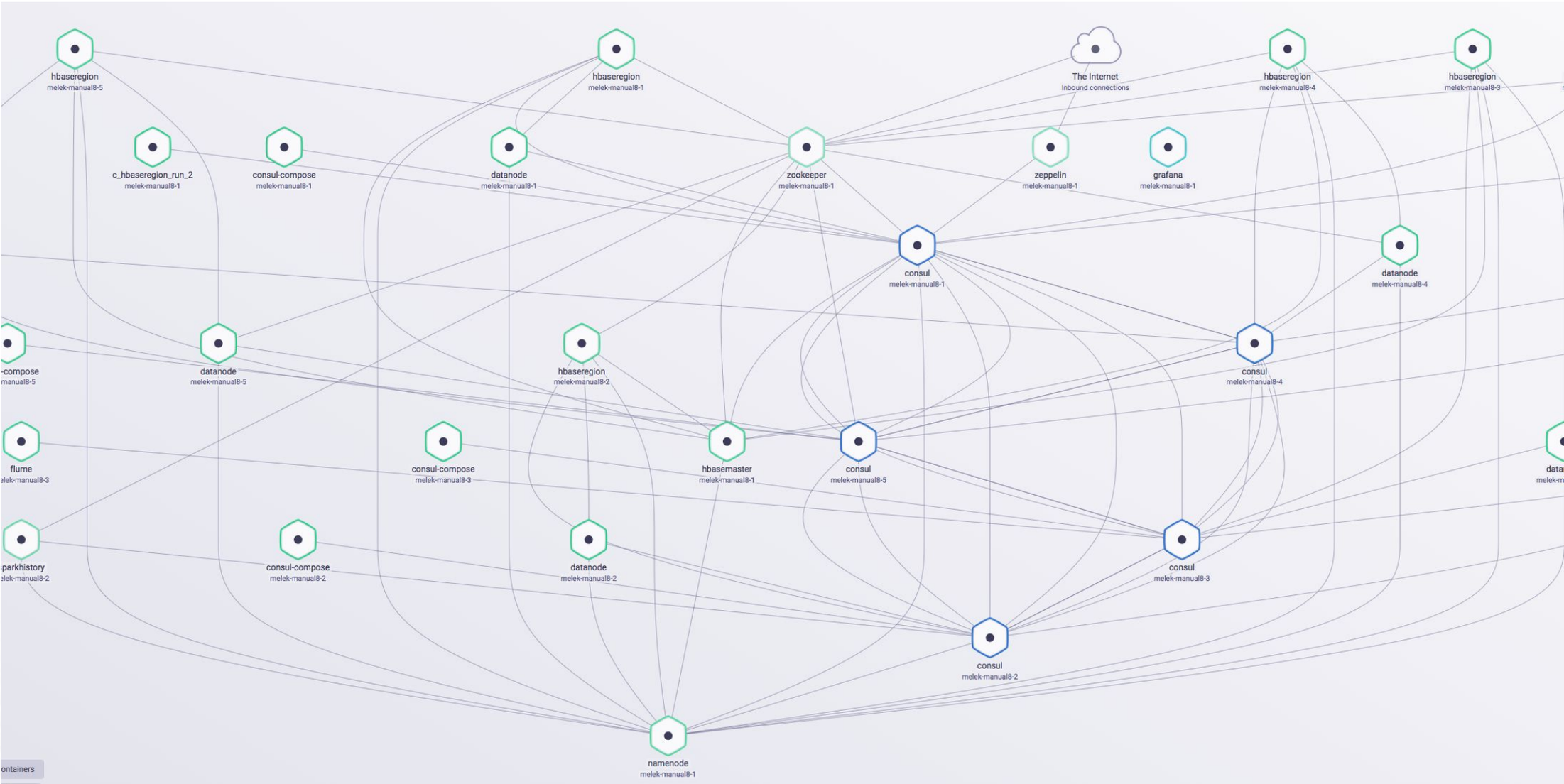
Microservice definition

”An approach to developing a single application as a

- suite of **small services**, each running in its own process
- and **communicating** with lightweight mechanisms, often an HTTP resource API.
- These services are built around business capabilities and **independently deployable** by fully automated deployment machinery.”

  – https://martinfowler.com/articles/microservices.html

Hadoop cluster



- The definition is almost true for a Hadoop cluster as well

Dockerized Hadoop cluster

- **How can we use the tools from microservice architecture in hadoop ecosystem?**
- A possible approach to install cluster (hadoop, spark, kafka, hive) based on
  - separated docker containers
  - Smart **configuration management** (using well-known tooling from microservices architectures)
- Goal: rapid prototyping platform
- Easy switch between
  - versions (official HDP, snapshot build, apache build)
  - configuration (ha, kerberos, metrics, htrace…)
- Developers/Ops tool
  - Easy != easy for any user without knowledge about the tool
- Not goal:
  - replace current management plaforms (eg. Ambari)

What are the Microservices (Theory)

12 Factory apps (http://12factor.net)

Collection of patterns/best practices

- **II. Dependencies**
  - Explicitly declare and isolate dependencies
- **III. Config**
  - Store config in the environment
- **VI. Processes**
  - Execute the app as one or more stateless processes
- **VIII. Concurrency**
  - Scale out via the process model
- **XII. Admin processes**
  - Run admin/management tasks as one-off processes

THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

BACKGROUND

The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the Heroku platform.

This document synthesizes all of our experience and observations on a wide variety of software-as-a-service apps in the wild. It is a triangulation on ideal practices for app development, paying particular attention to the dynamics of the organic growth of an app over time, the dynamics of collaboration between developers working on the app's codebase, and avoiding the cost of software erosion.

Our motivation is to raise awareness of some systemic problems we've seen in modern application development, to provide a shared vocabulary for discussing those problems, and to offer a set of broad conceptual solutions to those problems with accompanying terminology. The format is inspired by Martin Fowler's books *Patterns of Enterprise Application Architecture* and *Refactoring*.

WHO SHOULD READ THIS DOCUMENT?

Any developer building applications which run as a service. Ops engineers who deploy or manage such applications.

THE TWELVE FACTORS

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies

HORTONWORKS

What are the Microservices (Practice)

- Spring started as a
  - Dependency injection framework
- Spring Boot ecosystem
  - Easy to use starter projects
  - Lego bricks for various problems
    - JDBC access
    - Database access
    - REST
    - Health check

```
public class TimeStarter {


  @Autowired
  TimeService timerService;


  public Date now() {
      long timeService = timerService.now();
  }
}
```

- Spring Cloud -- elements to build microservices (based on Netflix stack)
  - **API gateway**
  - **Service registry**
  - **Configuration server**
  - Distributed tracing
  - Client side load balancing

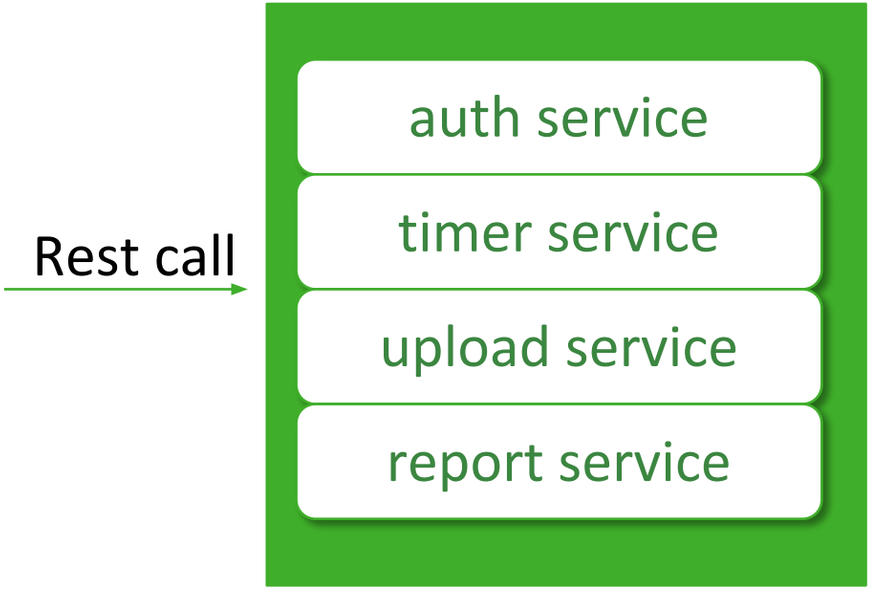
HORTONWORKS

Microservices with Spring Cloud

HORTONWORKS

Monolith application

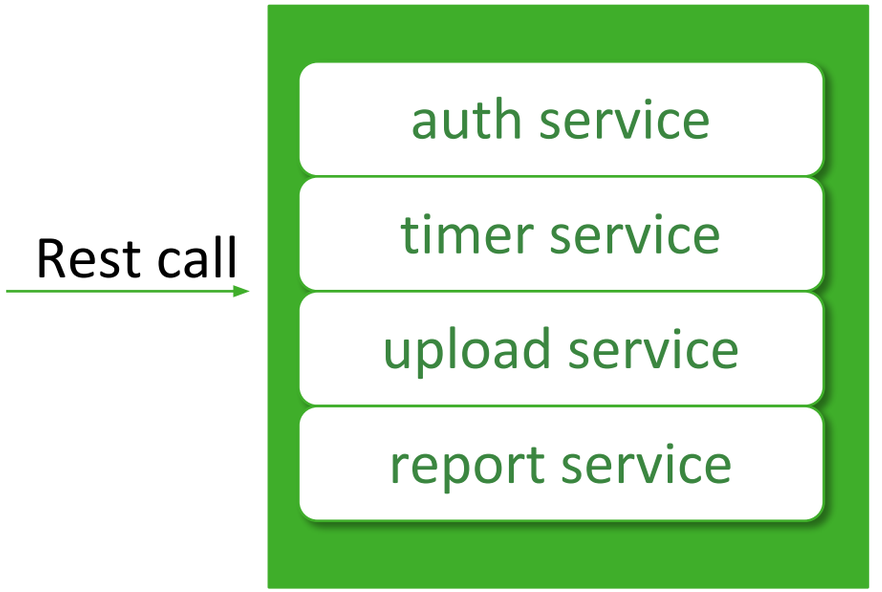
- Monolith but **modular** application example



Monolith application

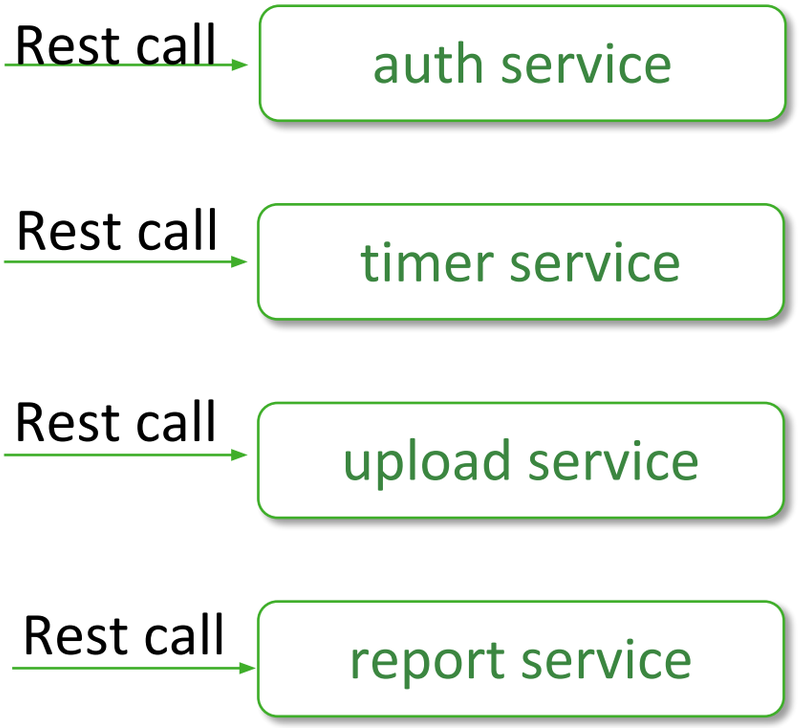
- Monolith but **modular** application example

```
@EnableAutoConfiguration
@RestController
@ComponentScan
public class TimeStarter {

  @Autowired
  TimeService timerService;

  @RequestMapping("/now")
  public Date now() {
      return timerService.now();
  }

  public static void main(String[] args) {
      SpringApplication.run(TimeStarter.class, args);
  }
}
```



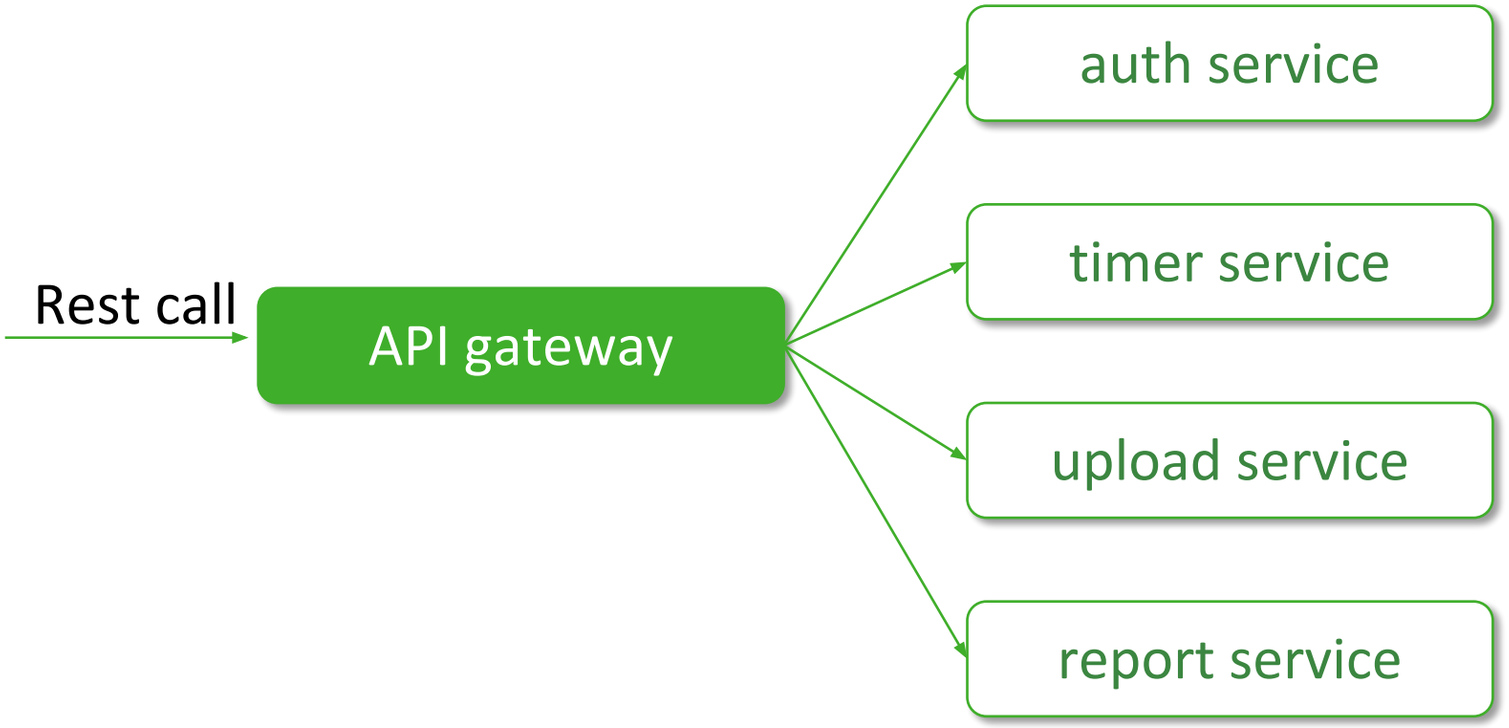
Microservice version

- First problem: how can we find the right backend port form the frontend?



Solution: API Gateway

- First problem: how can we find the right backend port form the frontend?

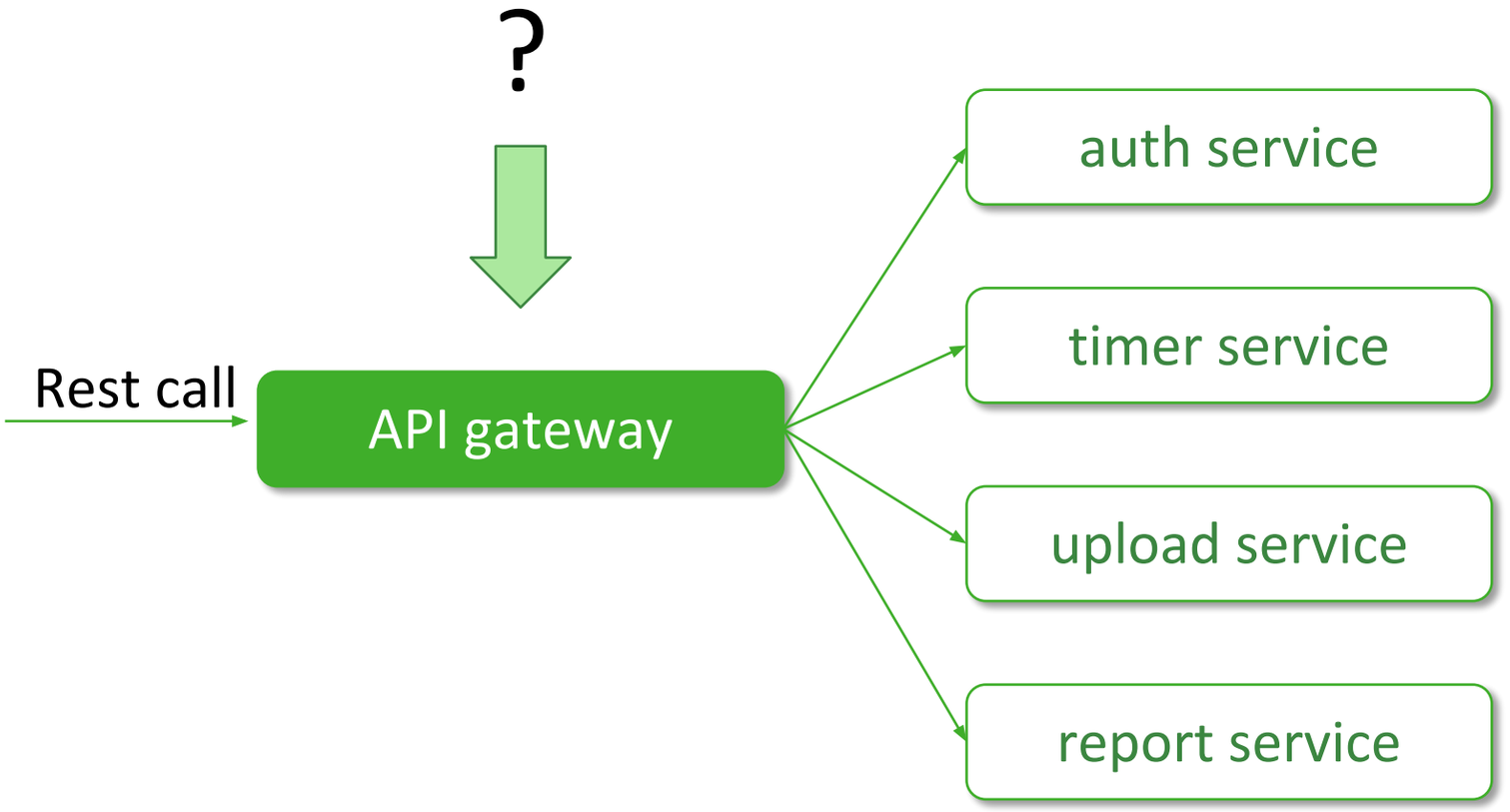


API Gateway

- **Goals:** Hide available microservices behind a service facade pattern
  - Routing, Authorization
  - Deployment handling, Canary testing, Blue/Green deployment
  - Logging, SLA, Auditing
- **Implementation examples:**
  - Spring cloud Api Gateway (based on Netflix Zuul)
  - Netflix Zuul based implementation
  - Twitter Finagle based implementation
  - Amazon API gateway
  - Simple Nginx reverse proxy configuration
  - Traefik, Kong
- Usage in Hadoop ecosystem
  - For prototyping: Only if the scheduler/orchestrator starts the service on a random host
  - For security: Apache Knox

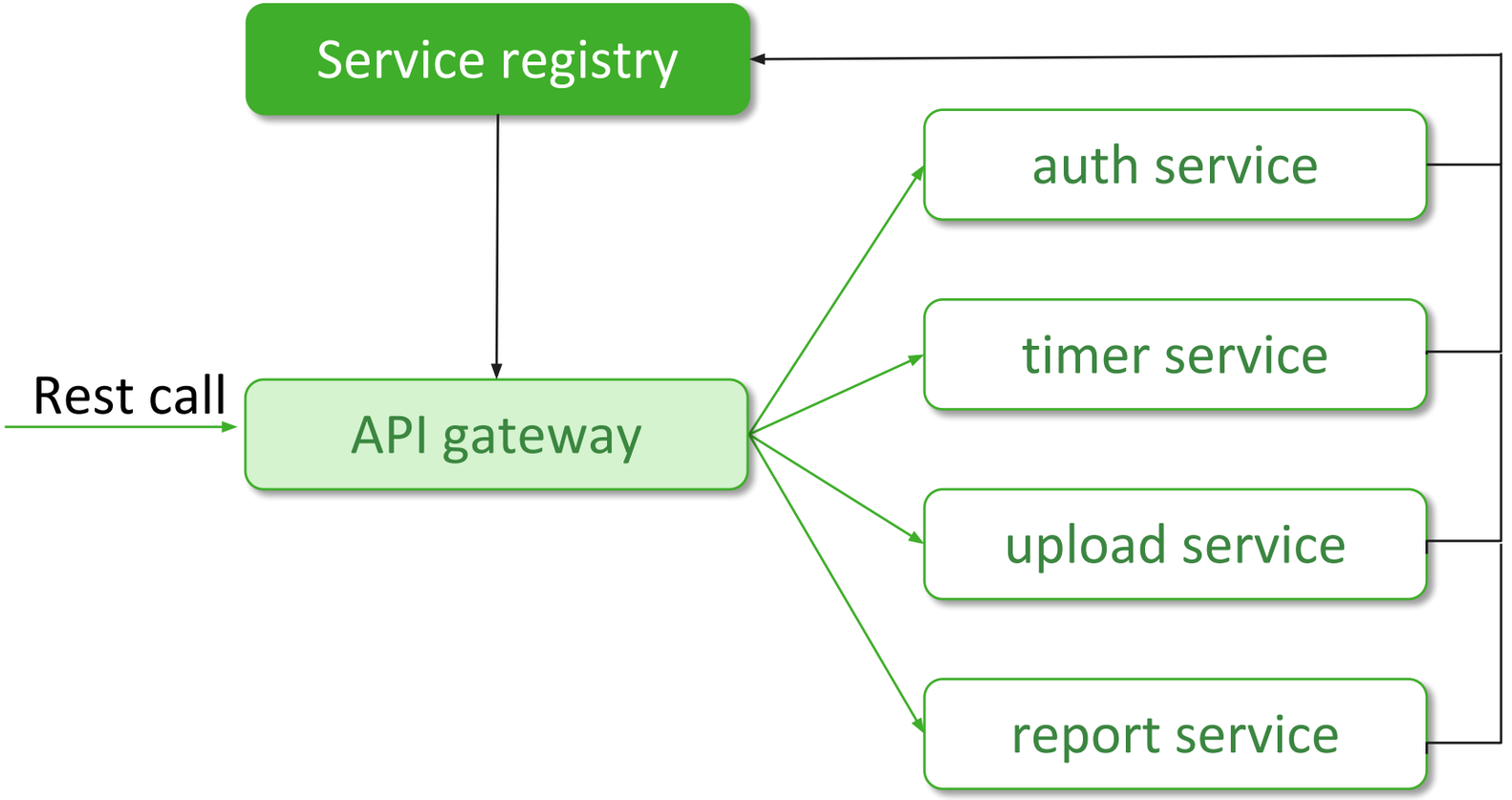
Service registry

- Problem: how to configure API gateway to automatically route to all the services



Service registry

- Solution: Use service registry
  - Components should be registered to the service registry automatically

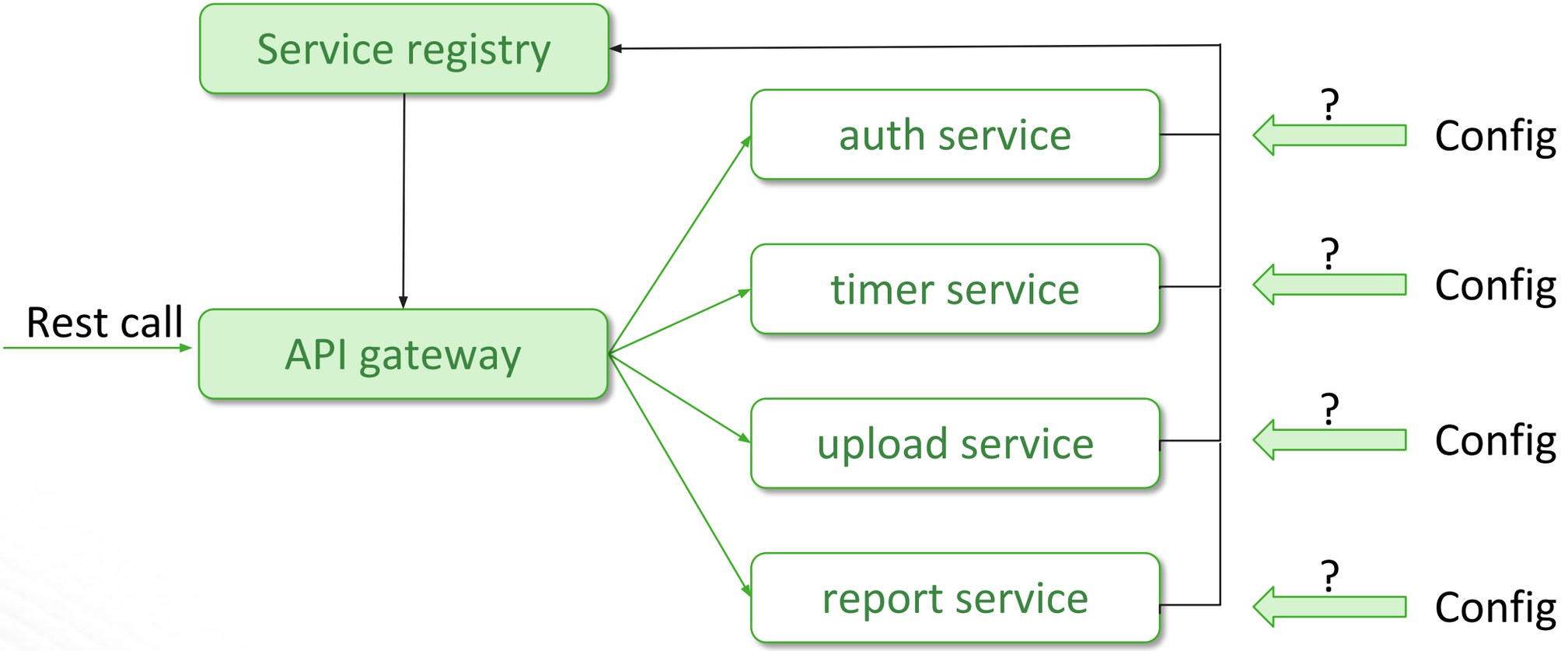


Service registry

- **Goal**: Store the location and state of the available services
  - Health check
  - DNS interface
- **Implementation examples**:
  - Spring cloud: Eureka
  - Netflix eureka based implementation
  - Consul.io
  - etcd, zookeeper
  - Simple workaround: DNS or hosts file
- Usage in Hadoop ecosystem
  - Most of the components needs info about the location of nameserver(s) and other *master* components

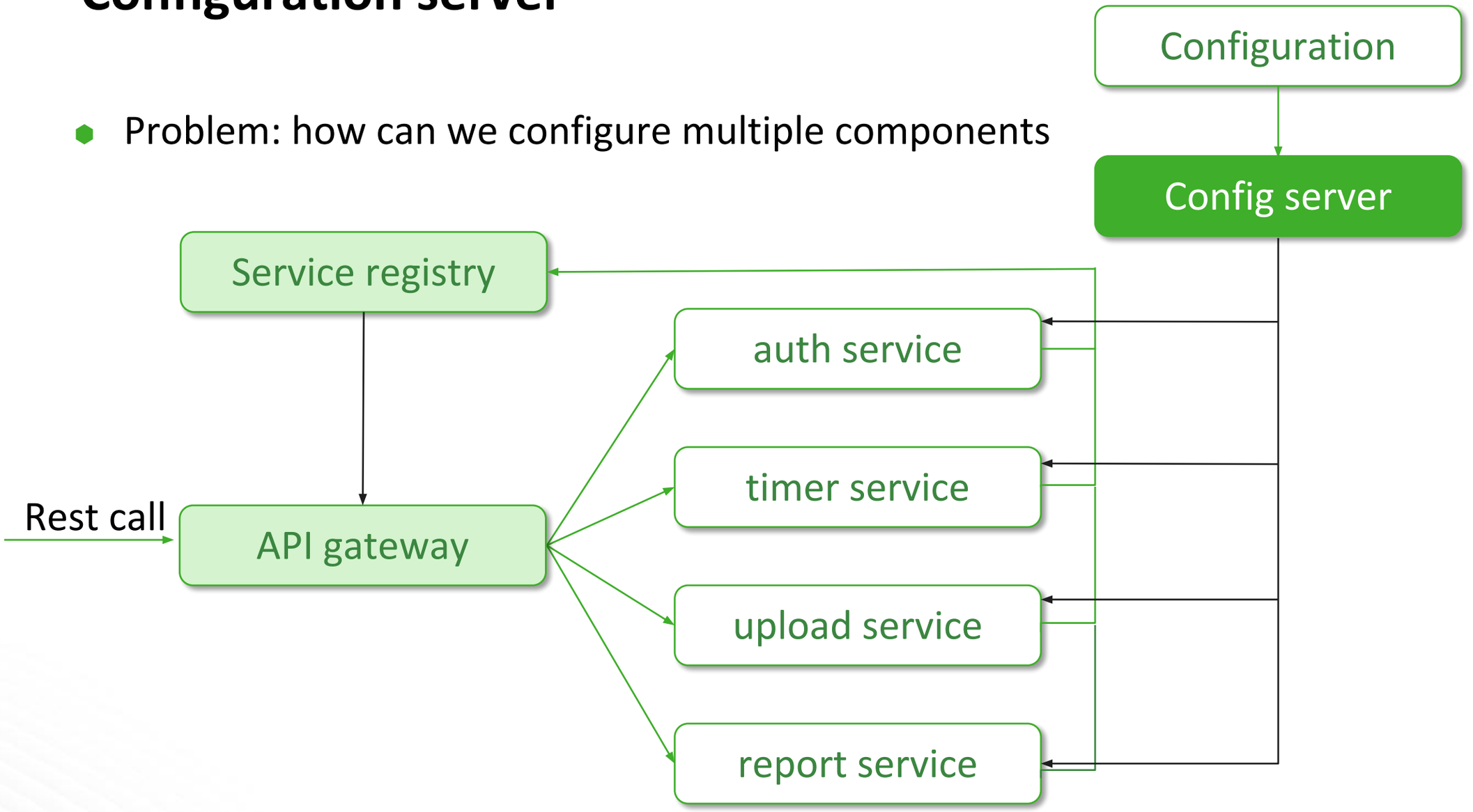
Configuration server

- Problem: how can we configure multiple components
  - ”Store config in the environment” (12factor)



Configuration server

- Problem: how can we configure multiple components

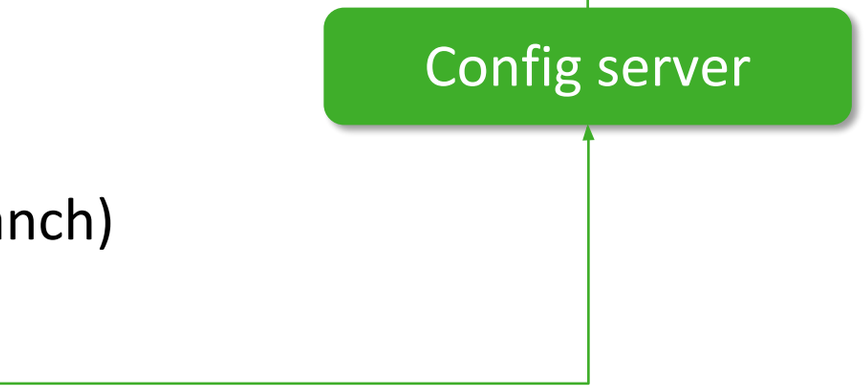


Config server

- Goals: One common place for all of the configuration
  - **Versioning**
  - Auditing
  - Multiple environment support: Use (almost) the same configuration from DEV to PROD environment
  - Solution for sensitive data
- **Solution examples**:
  - Spring Cloud config service
  - Zookeeper
  - Most of the service registry have key->value store (Consul, etcd)
  - Any persistence datastore (But the versioning is a question)
- For Hadoop ecosystem:
  - Most painful point: the same configuration elements (eg. core-site.xml) is needed at multiple location
  - Ambari and other management tools try to solve the problem (but not with the focus of rapid prototyping)

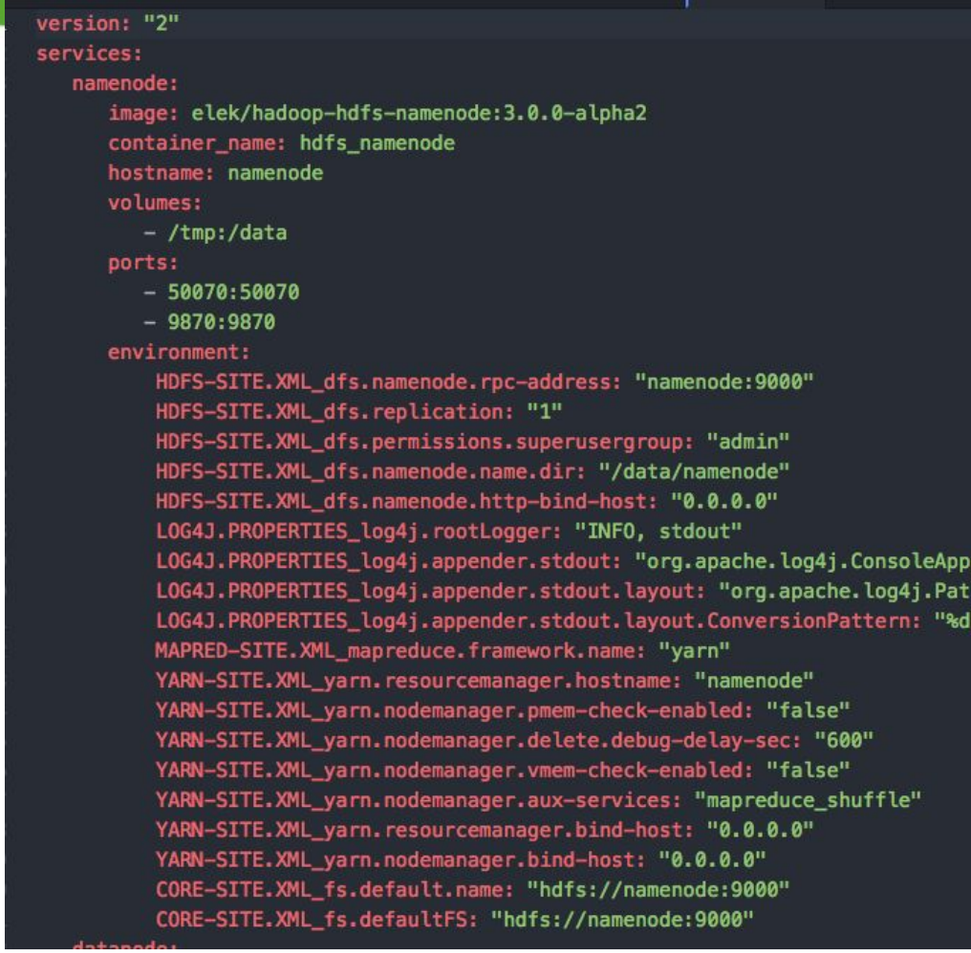
Config server – configuration management

- Config server structure: [branch]/name-profile.extension
- Merge properties for **name=timer** and **profile(environment)=dev**
- URL from the config server
  - http://config:8888/timer-dev.properties
    - server.port=6767
    - aws.secret.key=zzz
    - exit.code=-23



- Local file system structure (master branch)
  - **timer.properties**
    - server.port=6767
  - **dev.properties**
    - aws.secret.key=xxx
  - **application.properties**
    - exit.code=-23

Summary



```
version: "2"
services:
   namenode:
      image: elek/hadoop-hdfs-namenode:3.0.0-alpha2
      container_name: hdfs_namenode
      hostname: namenode
      volumes:
         - /tmp:/data
      ports:
         - 50070:50070
         - 9870:9870
      environment:
          HDFS-SITE.XML_dfs.namenode.rpc-address: "namenode:9000"
          HDFS-SITE.XML_dfs.replication: "1"
          HDFS-SITE.XML_dfs.permissions.superusergroup: "admin"
          HDFS-SITE.XML_dfs.namenode.name.dir: "/data/namenode"
          HDFS-SITE.XML_dfs.namenode.http-bind-host: "0.0.0.0"
          LOG4J.PROPERTIES_log4j.rootLogger: "INFO, stdout"
          LOG4J.PROPERTIES_log4j.appender.stdout: "org.apache.log4j.ConsoleApp
          LOG4J.PROPERTIES_log4j.appender.stdout.layout: "org.apache.log4j.Pat
          LOG4J.PROPERTIES_log4j.appender.stdout.layout.ConversionPattern: "%d
          MAPRED-SITE.XML_mapreduce.framework.name: "yarn"
          YARN-SITE.XML_yarn.resourcemanager.hostname: "namenode"
          YARN-SITE.XML_yarn.nodemanager.pmem-check-enabled: "false"
          YARN-SITE.XML_yarn.nodemanager.delete.debug-delay-sec: "600"
          YARN-SITE.XML_yarn.nodemanager.vmem-check-enabled: "false"
          YARN-SITE.XML_yarn.nodemanager.aux-services: "mapreduce_shuffle"
          YARN-SITE.XML_yarn.resourcemanager.bind-host: "0.0.0.0"
          YARN-SITE.XML_yarn.nodemanager.bind-host: "0.0.0.0"
          CORE-SITE.XML_fs.default.name: "hdfs://namenode:9000"
          CORE-SITE.XML_fs.defaultFS: "hdfs://namenode:9000"
```

- Tools used in microservice architecture
- Key components:
  - Config server
  - Service registry
  - API gateway
- Configuration server
  - Versioning
  - One common place to distribute configuration
  - **Configuration preprocessing!!!**
    - transformation
    - the **content** of the configuration should be defined, it could be format independent
    - But the final configuration should be visible

HORTONWORKS

Docker based Hadoop cluster

HORTONWORKS

How to do it with Hadoop?

apache-hadoop-X.X.tar.gz

- bin
  - hdfs
  - yarn
  - mapred
- etc/hadoop
  - **core-site.xml**
  - **mapred-site.xml**
  - **hdfs-site.xml**
- include
- lib
- libexec
- sbin
- share

Microservice architecture elements

1. **Configuration server**
2. **Service registry**
3. API gatway

HORTONWORKS

Do it with Hadoop

apache-hadoop-X.X.tar.gz

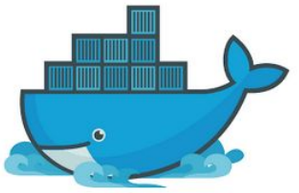
- bin
  - hdfs
  - yarn
  - mapred
- etc/hadoop
  - core-site.xml
  - mapred-site.xml
  - hdfs-site.xml
- include
- lib
- libexec
- sbin
- share

Microservice architecture elements

1. **Configuration server**
2. **Service registry**
3. API gatway
4. **+1 Packaging**

HORTONWORKS

Packaging: Docker



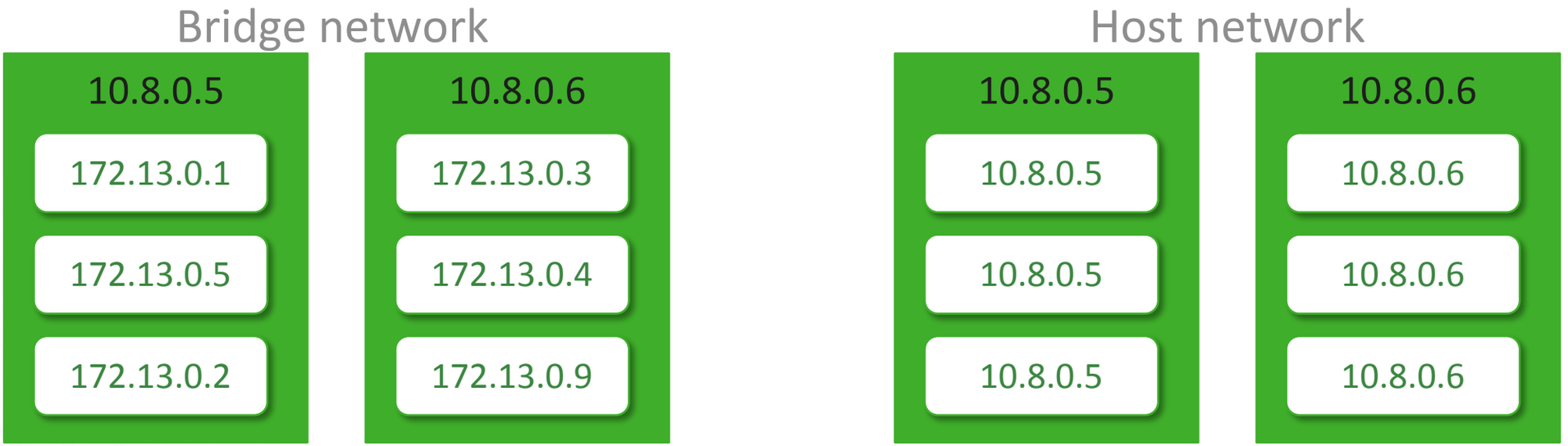
- Packaging: Docker
  - **Docker** Engine:
    - a portable,
    - lightweight runtime and
    - packaging tool
  - **Docker** Hub,
    - a cloud service for sharing applications
  - **Docker Compose:**
    - Predefined recipes (environment variables, network, …)

- My docker containers: http://hub.docker.com/elek/

HORTONWORKS

Docker decisions

- One application per container
  - More flexible
  - More simple (configuration preprocess + start)
  - One deployable unit
- Microservice-like: prefer more similar units against smaller but bigger one
- Using host network for clusters



Repositories

- elek/bigdata-docker:
  - example configuration
  - docker-compose files
  - ansible scripts
  - ***getting started*** **entrypoint**

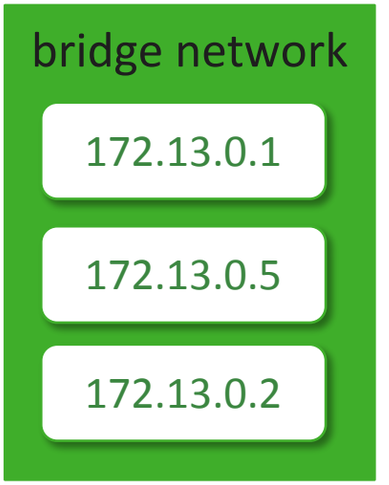
directory	configuration type	docker container starter	provisioning	cluster type	network (*)
simple	environment variables	docker-compose		local	Using host network
compose	environment variables	docker-compose		local	Using dedicated docker network
consul	consul	consul-composer (docker-compose)	consul-compose(+ansible)	local/cluster	Using host network
spring	spring config server	docker-compose		local/cluster	Using host network
ansible	environment variables	docker (ansible module)	ansible	cluster	Using host network

- elek/docker-bigdata-base (base image for all the containers)
  - Contains all the configuration loading (and some documentation)
  - Use CONFIG_TYPE environment variable to select configuration method
    - CONFIG_TYPE=simple (configuration from environment variables – for local env)
    - CONFIG_TYPE=consul (configuration from consul – for distributed environment)

- elek/docker-…. (hadoop/spark/hive/...)
  - Docker images for the components

Local demo

- Local run, using host network
  - More configuration is needed
  - **Auto scaling is supported**
  - https://github.com/elek/bigdata-docker/tree/master/compose



HORTONWORKS

Do it with Hadoop

apache-hadoop-X.X.tar.gz

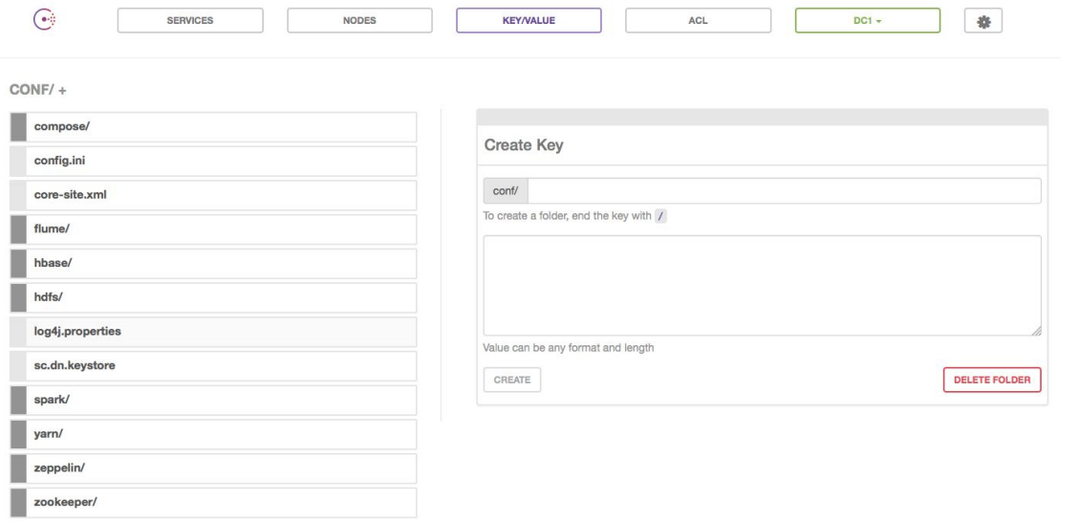
- bin
  - hdfs
  - yarn
  - mapred
- etc/hadoop
  - core-site.xml
  - mapred-site.xml
  - hdfs-site.xml
- include
- lib
- libexec
- sbin
- share

Components

1. **Packaging**
2. **Configuration server**
3. **Service registry**
4. API gateway

HORTONWORKS

Service registry/configuration server

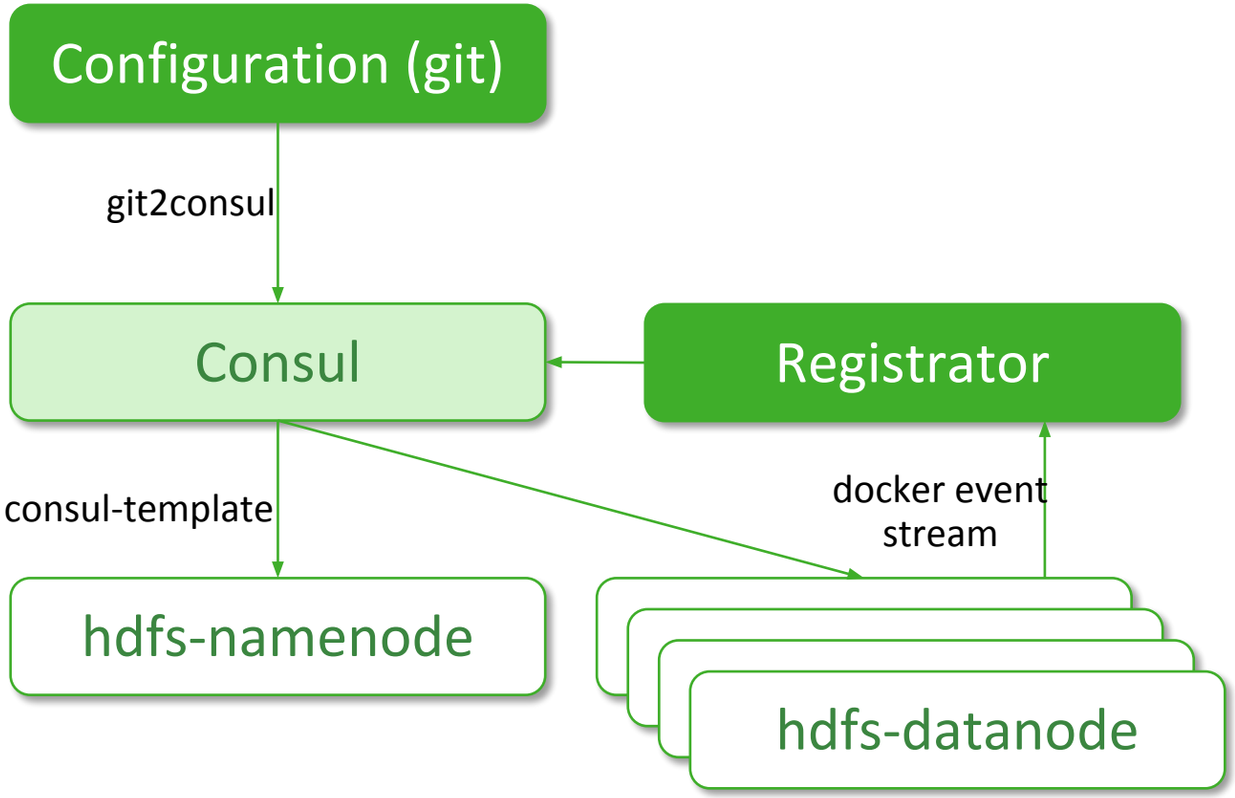


- Service registry
  - Health check support
  - DNS support
- Key-value store
  - Binary data is supported
- Based on agents and servers
- Easy to use REST API
- RAFT based consensus protocol

```
import consul

c = consul.Consul()

# poll a key for updates
index = None
while True:
    index, data = c.kv.get('foo', index=index)
    print data['Value']

# in another process
c.kv.put('foo', 'bar')
```

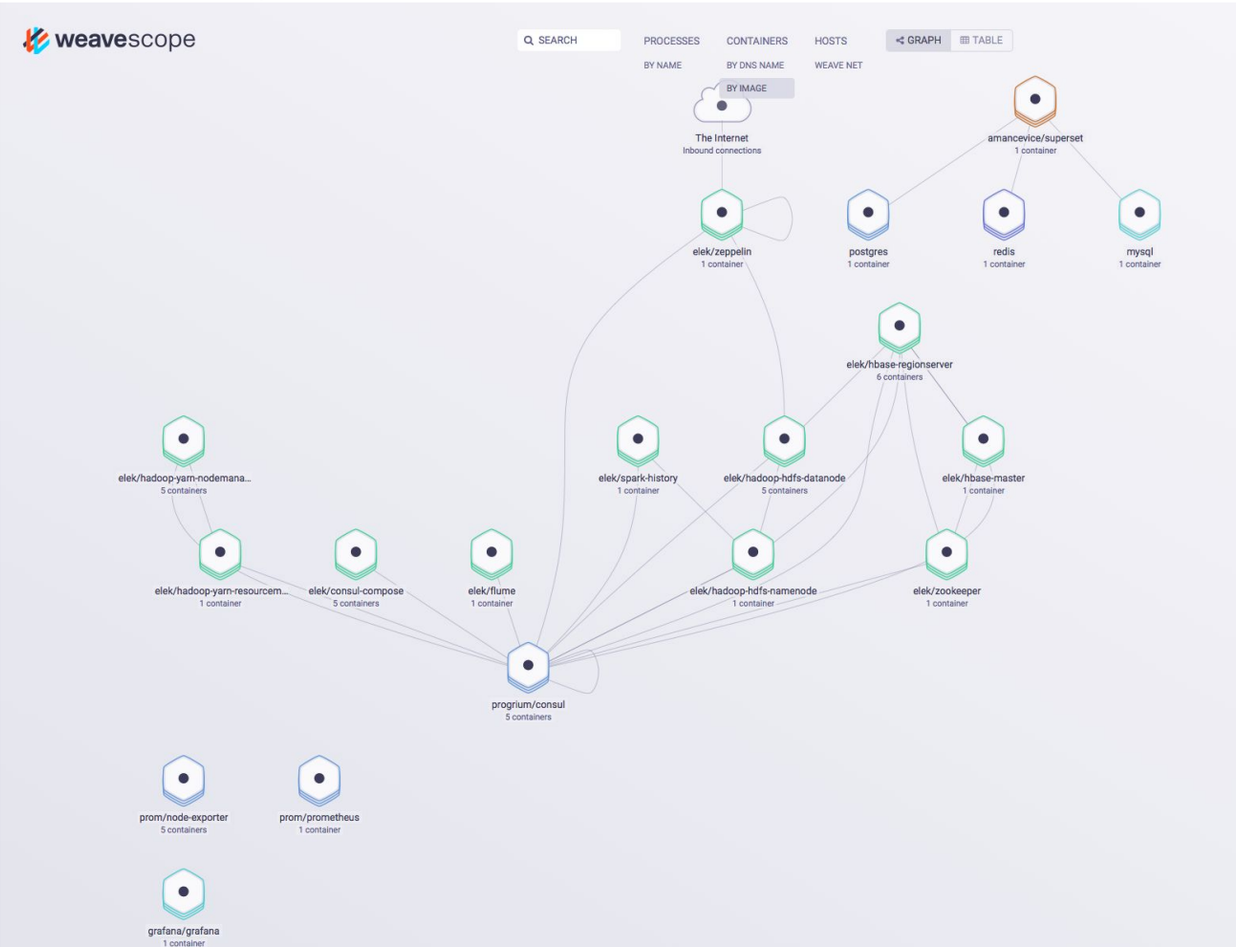
Service registry/configuration server

- Git2Consul
  - Mirror git repositories to consul
- Consul template
  - Advanced Template engine
  - Renders a template (configuration file) based on the information from the consul
  - Run/restart a process on change
- Registrator
  - Listen on docker event stream
  - Register new components to consul



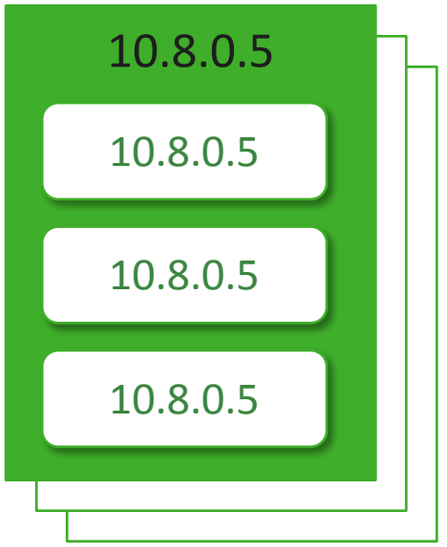
Weave scope

- Agents to monitor
  - network connections between components
  - cpu
  - memory
- Supports Docker, Swarm, Weave network, …
- Easy install
- Transparent
- Pluggable
- Only problems:
  - Temporary docker containers



Distributed demo

- Distributed run with host network
  - https://github.com/elek/bigdata-docker/tree/master/consul
  - Configuration is hosted in a consul instance
  - Dynamic update



TODO

- More profiles and configuration set
  - Ready to use kerberos/HA environments
  - On the fly keytab/keystore generation (?)
- Scripting/tool improvement
  - Autorestart in case of service registration change
- Configuration for more orcherstration/scheduling
  - Nomad?
  - Docker Swarm?
- Easy image creation for specific builds
- Improve docker images
  - Predefined volume/port definition
  - Consolidate default values

Thank You