

微服务架构之服务发现

Wen Zhenglin
2016-6-28

Overview

Service Discovery introduction

Comparison

Introduce Consul

Service discovery example

Some thought

Summary

Background

We have already experience the distributed app in many place(server), we almost have no manage for it, just ad hoc

We are starting to implement our business in microservice architecture, Since it may lead to so many service, it becomes even more challenge, hard to maintain, and error-prone for human operation, ad hoc manage is not working anymore

Here we are trying to find out a way to manage our so many app/services, to avoid the cost of chaos maintenance, and make our life easier later

Service manage

Service manage at least include the following core part:

- Service discovery
- Service monitoring

We will primary talking about service discovery here

Service Discovery introduction

What is a Service

Service is independently deployable, small (in business scope) and modular

Service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goal

Why service discovery

Service discovery (ip and port may dynamic change, or we actively do the change) (say 100 service use service-A, what if service-A change its ip)

Also we want to know all the service out there, to see them, and the basic health status

We often using service registry (to register it first, so each service can discover each others later)

About service discovery

Service discovery is not means to abandon parameter or env setting

We should have alternative method (for maintenance or temporary solution, eg. service registry is down)

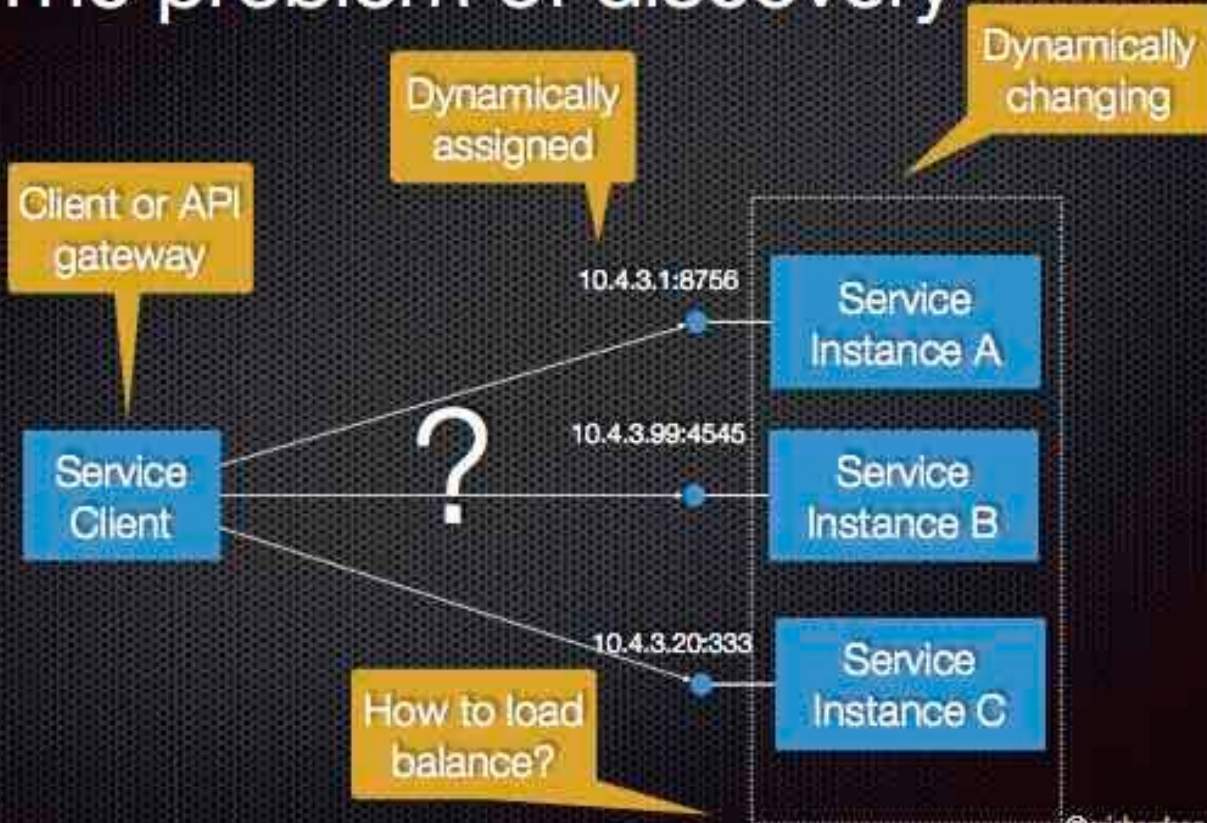
Discovery Pattern

The problem of discovery

Client-side discovery

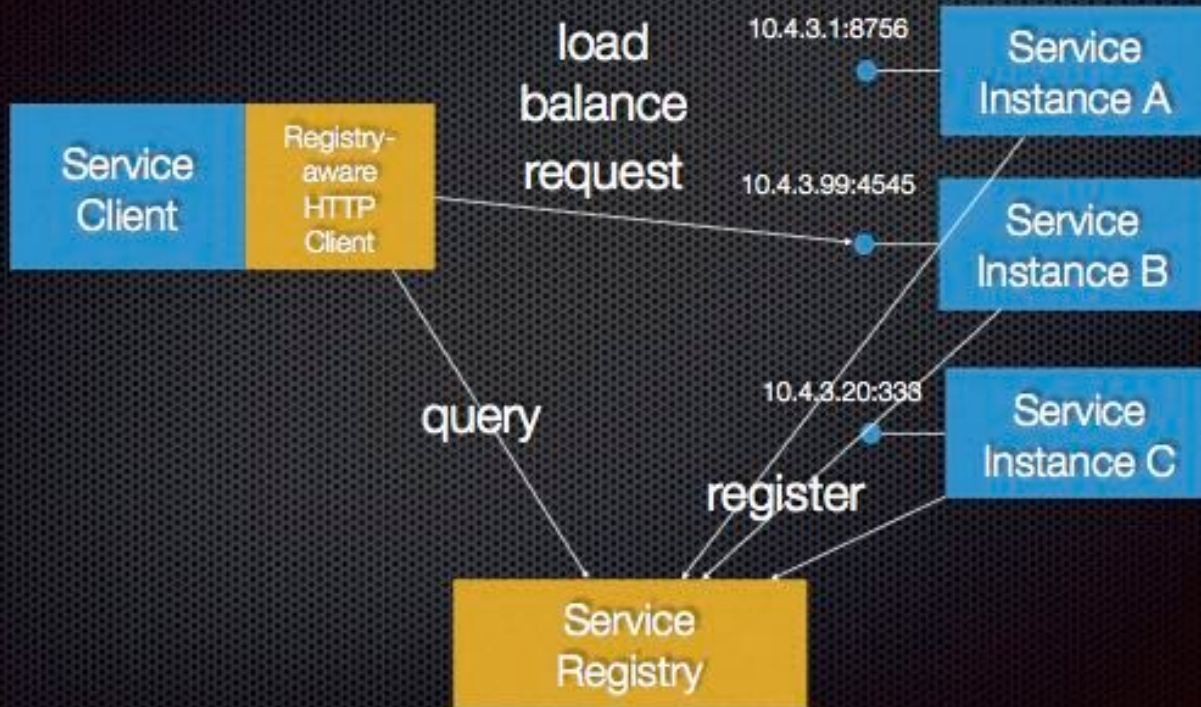
Server-side discovery

The problem of discovery



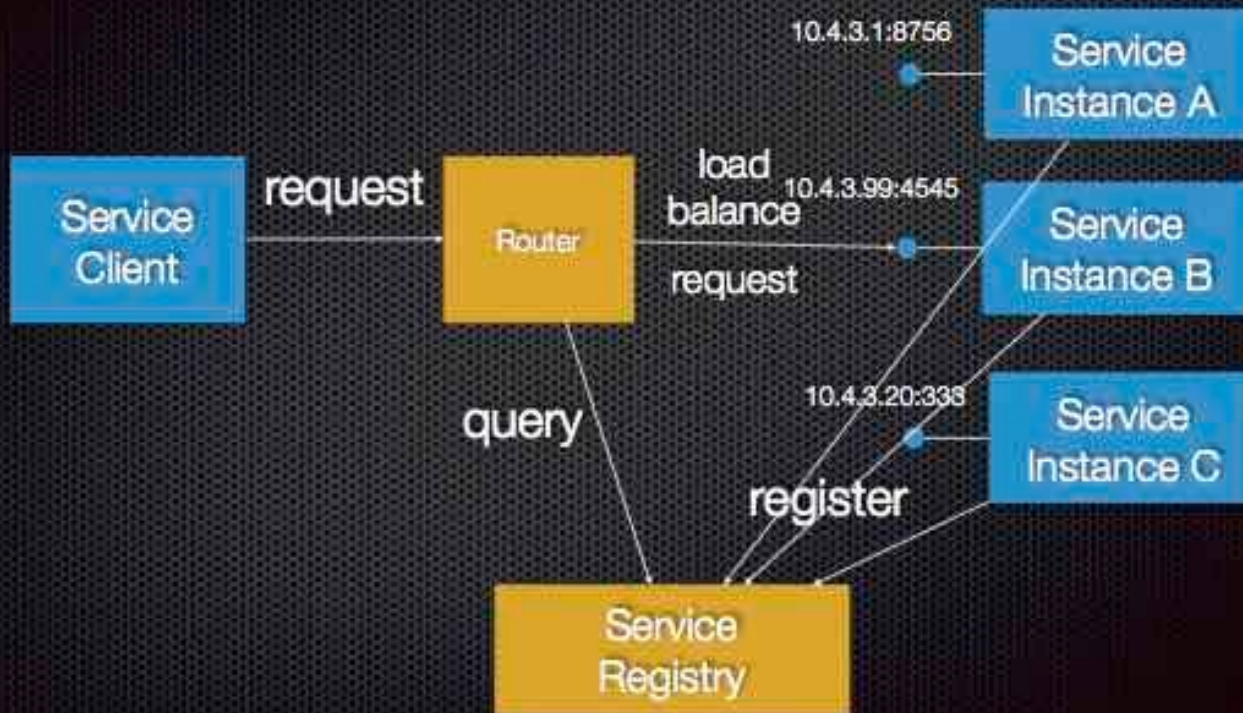
2

Pattern: Client-side discovery



3

Pattern: Server-side discovery



About the router

Even though the client code is simpler, the router is another system component that must be installed and configured. It will also need to be replicated for availability and capacity

Introduce extra bond (many service bond to a router)

Its both need doing a request (our discovery is a simple request only)

For now we just simply use client side discovery

Comparison

Basic requirement for the solution

We so much heavy depend on it, it must satisfy the following:

- High Availability (fault-tolerance, often a distributed system of cluster)

Distributed system need to have a consensus mechanism

It's hard to implement it right (for distributed system)

Three existing solution

Zookeeper

Etcd

Consul

Comparison-1

Area/Name	Zookeeper	Etcd	Consul
Service concept	no	no	yes
Multiple datacenter	no	no	yes
User interface	no	no	yes
Key-value store	yes	yes	yes
Event watching	yes	yes	yes
DNS support	no	no	yes
Health monitoring	no	no	yes
Polyglot (多语种) support	Java-only + 3rd-party	Go-only + 3rd-party	<u>yes</u>

Comparison-2

Area/Name	Zookeeper	Etcd	Consul
Written in	Java	Go	Go
HTTP-API	no	yes	yes
Client-API	no	http-json	http-json, config-file, dns
CLI	zkCli	etcd-ctl	consul-cli
CAP type	AP	AP	CP
Consensus algorithm	paxos-variant	raft	raft
Vendor	Apache	CoreOS	Hashicorp
License	Apache License 2.0	Apache License 2.0	Mozilla Public License, version 2.0

Relate ecosystem

Zookeeper -- Apache, Kafka, Storm, Hadoop, etc

Etcd -- CoreOS, rkt, fleet, Kubernetes, etc

Consul -- HashiCorp, Vagrant, Packer, Terraform, etc

Technical term explain

Paxos

Zab

Raft

Gossip

Fault-tolerance and consensus

It is well known that fault-tolerance on commodity hardware can be achieved through replication

A common approach is to use a consensus algorithm to ensure that all replicas are mutually consistent

By repeatedly applying such an algorithm on a sequence of input values, it is possible to build an identical log of values on each replica

Consensus algorithm usage

System	Protocol	Implementation	Usage
Google GFS	Multi-Paxos	Chubby	Lock Service
Google Spanner	Multi-Paxos	Chubby	
Google Borg	Multi-Paxos	Chubby	Configuration, Master election
Apache HDFS	Zab	ZooKeeper	Failure detection, Active NameNode election
Apache Giraph	Zab	ZooKeeper	Coordination, Configuration, Aggregators
Apache Hama	Zab	ZooKeeper	Coordination
CoreOS	Raft	etcd	Service Discovery
OpenStack	Zab	ZooKeeper	Service Discovery
Apache Kafka	Zab	ZooKeeper	Coordination, Configuration
Apache BookKeeper	Zab	ZooKeeper	Coordination, Configuration

Source: <http://metadata156.rssing.com/chan-6816089/latest.php> (2015-10-5)

Paxos

Paxos is a family of protocols for solving consensus in a network of unreliable processors.

Consensus is the process of agreeing on one result among a group of participants

Paxos (cont.)

Paxos has dominated the discussion of consensus algorithms over the last decade: most implementations of consensus are based on Paxos or influenced by it, and Paxos has become the primary vehicle used to teach students about consensus

Unfortunately, Paxos is quite difficult to understand, in spite of numerous attempts to make it more approachable. Furthermore, its architecture requires complex changes to support practical systems. As a result, both system builders and students struggle with Paxos

Zookeeper atomic broadcast (ZAB)

Zab looks like Multi-Paxos, with a leader and unique proposals and majority acknowledgements for commits

Zab follows strict ordering decided by the Leader. If the leader goes down, the next leader is not allowed to reorder requests, unlike Multi-Paxos

Zab is more useful for backup systems, whereas Multi-Paxos is more suited for state replication

Raft

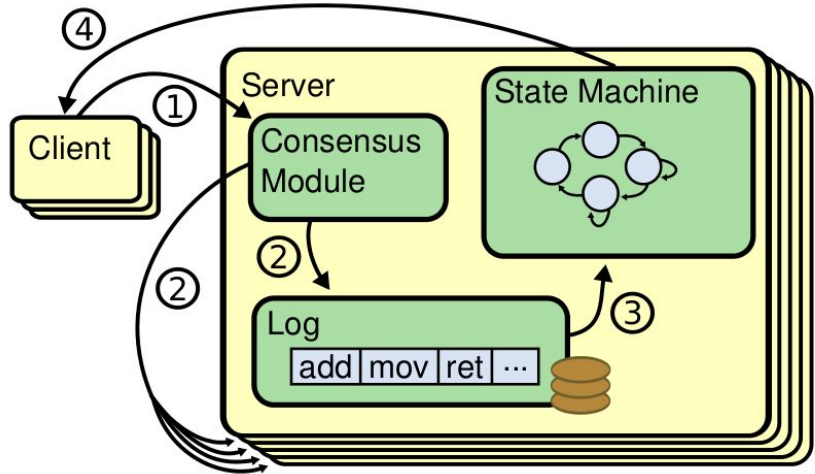
Raft is a consensus algorithm that is designed to be easy to understand. It's equivalent to Paxos in fault-tolerance and performance

The difference is that it's decomposed into relatively independent subproblems, and it cleanly addresses all major pieces needed for practical systems

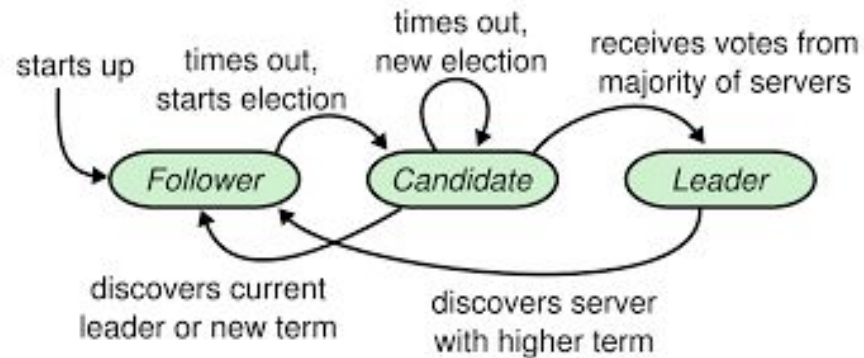
<https://raft.github.io/>

<http://thesecretlivesofdata.com/raft/>

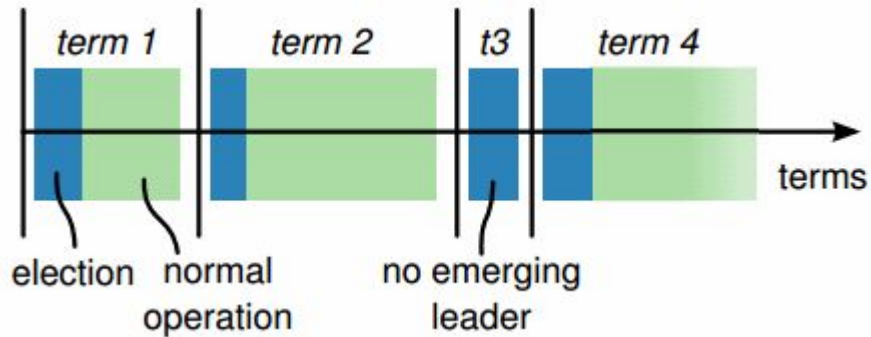
Raft (cont.)



Replicated state machine

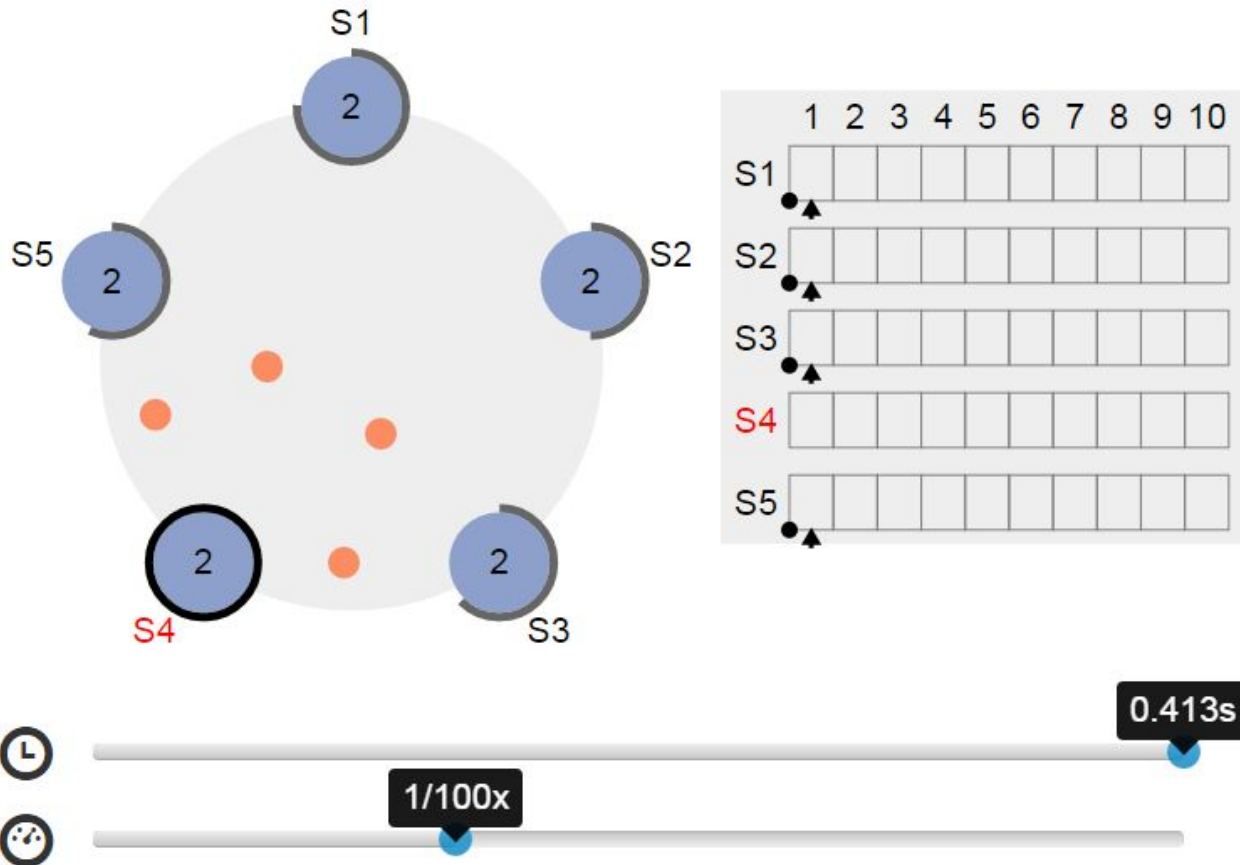


Server states



Time is divided into terms

Raft Visualization



Gossip

Consul uses a [gossip protocol](#) to manage membership and broadcast messages to the cluster, provided by the [Serf library](#)

The gossip protocol used by Serf is based on "[SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol](#)", with a few minor adaptations

Gossip (cont.)

A gossip protocol is a style of computer-to-computer [communication protocol](#) inspired by the form of [gossip](#) seen in social networks

Modern [distributed systems](#) often use gossip protocols to solve problems that might be difficult to solve in other ways, either because

- the underlying network has an inconvenient structure
- is extremely large
- or because gossip solutions are the most efficient ones available

Zookeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services

All of these kinds of services are used in some form or another by distributed applications

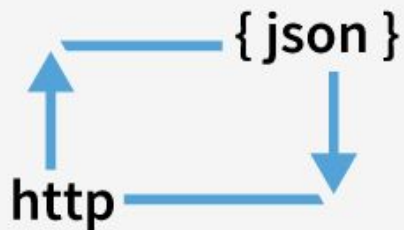
Zookeeper (cont.)

- Name Service, Configuration, Group Membership (key-value store)
- Barriers (watch event with barrier node)
- Queues (why not use kafka,nsq instead)
 - the new pathnames will have the form `_path-to-queue-node_/queue-X`, here X is a monotonic increasing number
- Locks
 - Call `create()` with a pathname of `"_locknode_/lock-"` and the sequence and ephemeral flags set
- Leader Election (Watch for changes on `"ELECTION/n_j"`, where j is the largest sequence number such that $j < i$ and `n_j` is a znode in C)

<https://zookeeper.apache.org/doc/r3.4.8/recipes.html> 2016-2-21

demo

A distributed, reliable key-value store for the most critical data of a distributed system



Simple Interface

Read and write values with curl
and other HTTP libraries

```
/config
├ /database
/feature-flags
├ /verbose-logging
└ /redesign
```

Key/Value Storage

Store data in directories, similar
to a file system



Watch for Changes

Watch a key or directory for
changes and react to the new
values

Etd examples

```
$ etcdctl set /message Hello  
Hello
```

```
$ curl -L -X PUT http://127.0.0.1:2379/v2/keys/message -d value="Hello"  
{  
  "action": "set",  
  "node": {  
    "key": "/message",  
    "value": "Hello",  
    "modifiedIndex": 4,  
    "createdIndex": 4  
  }  
}
```

```
$ etcdctl get /message  
Hello
```

```
$ curl -L http://127.0.0.1:2379/v2/keys/message  
{  
  "action": "get",  
  "node": {  
    "key": "/message",  
    "value": "Hello",  
    "modifiedIndex": 4,  
    "createdIndex": 4  
  }  
}
```

demo

Introduce Consul



[Intro](#) [Docs](#) [Community](#) [Demo](#) [Download](#) [GitHub](#)

Service discovery and
configuration made easy.
Distributed, highly
available, and datacenter-
aware.



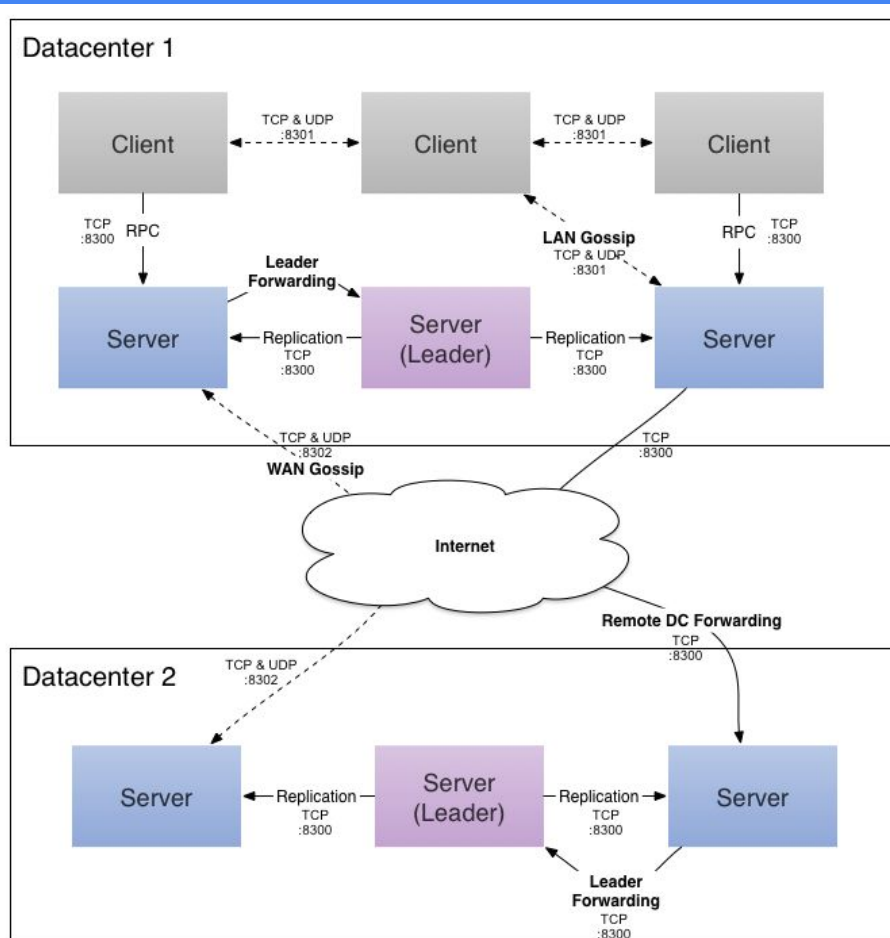
Consul (cont1.)

- Service Discovery
 - Consul makes it simple for services to register themselves and to discover other services via a DNS or HTTP interface. Register external services such as SaaS providers as well
- Failure Detection
 - Pairing service discovery with health checking prevents routing requests to unhealthy hosts and enables services to easily provide circuit breakers

Consul (cont2.)

- Multi Datacenter
 - Consul scales to multiple datacenters out of the box with no complicated configuration. Look up services in other datacenters, or keep the request local
- Key/Value Storage
 - Flexible key/value store for dynamic configuration, feature flagging, coordination, leader election and more. Long poll for near-instant notification of configuration changes

Consul architecture



https://demo.consul.io/ui/#/nyc3/services/web

SERVICES NODES KEY/VALUE ACL NYC3

Filter by name any status EXPAND

consul	9 passing
proxy	4 passing
redis	12 passing
web	12 passing

web

TAGS
No tags

NODES

consul-client-nyc3-1	104.236.227.127	4 passing
Disk Util	Disk Util	passing
Load Avg	Load Avg	passing
Serf Health Status	serfHealth	passing
Service 'web' check	service:web	passing
consul-client-nyc3-2	104.236.229.68	4 passing
Disk Util	Disk Util	passing
Load Avg	Load Avg	passing
Serf Health Status	serfHealth	passing

demo

Service

The agent provides a simple service definition format to declare the availability of a service

And to potentially associate it with a health check

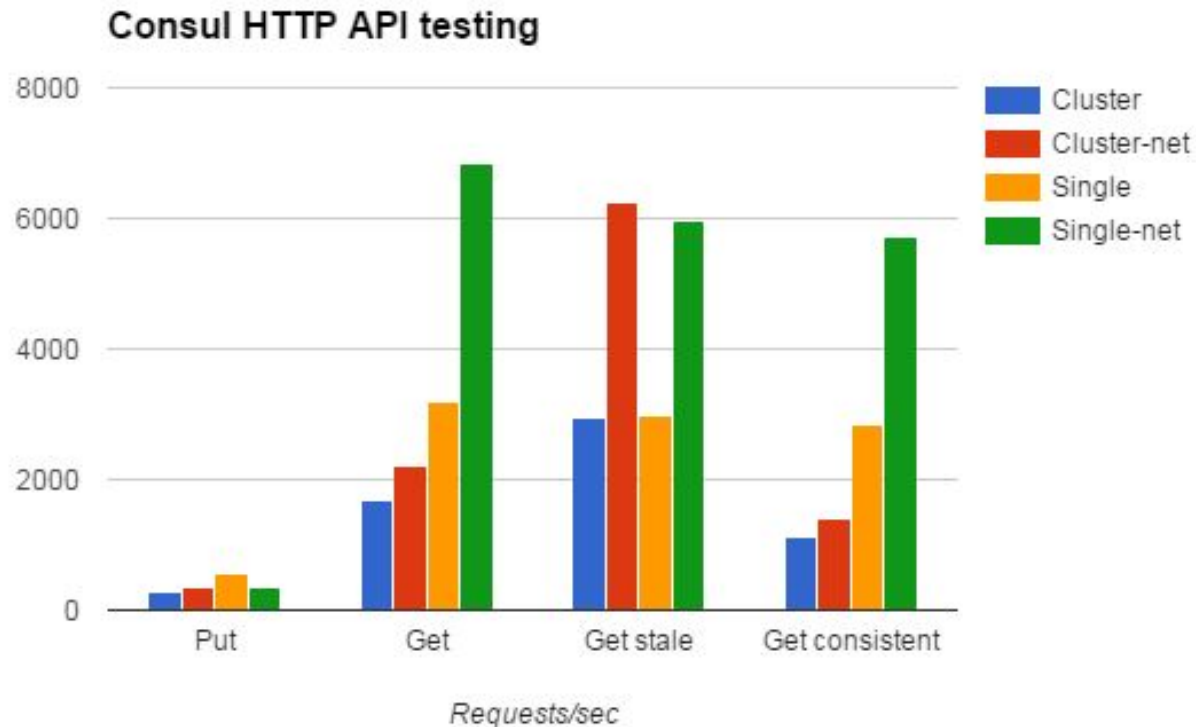
Consul exposes service definitions and tags over the [DNS](#) interface

```
{
  "service": {
    "name": "redis",
    "tags": ["master"],
    "address": "127.0.0.1",
    "port": 8000,
    "enableTagOverride": false,
    "checks": [
      {
        "script": "/usr/local/bin/check_redis.py",
        "interval": "10s"
      }
    ]
  }
}
```

Key-value

The KV endpoints are used to access Consul's simple key/value store, useful for storing service configuration or other metadata

- [/v1/kv/<key>](#): Manages updates of individual keys, deletes of individual keys or key prefixes, and fetches of individual keys or key prefixes
- [/v1/txn](#): Manages updates or fetches of multiple keys inside a single, atomic transaction



Health checks

- [/v1/health/node/<node>](#): Returns the health info of a node
- [/v1/health/checks/<service>](#): Returns the checks of a service
- [/v1/health/service/<service>](#): Returns the nodes and health info of a service
- [/v1/health/state/<state>](#): Returns the checks in a given state

DNS support

- Node Lookups
 - `<node>.node[.datacenter].<domain>`
- Service Lookups
 - `[tag.]<service>.service[.datacenter].<domain>`
 - Passing service return only
- Prepared Query Lookups
 - `<query or name>.query[.datacenter].<domain>`
 - Can filtering by multiple tags and automatically failing over

To allow for simple load balancing, the set of nodes returned is randomized each time. Both A and SRV records are supported

HTTP-API interface

The main interface to Consul is a RESTful HTTP API. The API can be used to perform CRUD operations on nodes, services, checks, configuration, and more

- [acl](#) - Access Control Lists
- [agent](#) - Consul Agent
- [catalog](#) - Nodes and services
- [coordinate](#) - Network coordinates
- [event](#) - User Events
- [health](#) - Health checks
- [kv](#) - Key/Value store
- [query](#) - Prepared Queries
- [session](#) - Sessions
- [status](#) - Consul system status

Blocking Queries (same as watch)

A blocking query is used to wait for a potential change using long polling

Any endpoint that supports blocking will also set the HTTP header **X-Consul-Index**, a unique identifier representing the current state of the requested resource

Client can set the Index query string parameter to the value of **X-Consul-Index**, indicating it wishes to wait for any changes subsequent to that index

A **wait** parameter specifying a maximum duration for the blocking request

Consistency Modes

- **default** - If not specified, the default is strongly consistent in almost all cases.
- **consistent** - This mode is strongly consistent without caveats
- **stale** - This mode allows any server to service the read regardless of whether it is the leader

To switch these modes, either the stale or consistent query parameters should be provided on requests

Service discovery

HTTP

DNS

Service discovery by HTTP

/v1/agent/services -->

/v1/catalog/service/<service>

/v1/health/service/<service>?passing

```
{  
  "redis": {  
    "ID": "redis",  
    "Service": "redis",  
    "Tags": null,  
    "Address": "",  
    "Port": 8000  
  }  
}
```

Service discovery by DNS

```
$ dig @192.168.100.94 -p 8600 dns.service.consul
```

```
:: QUESTION SECTION:
```

```
;dns.service.consul.      IN      A
```

```
:: ANSWER SECTION:
```

```
dns.service.consul.      0       IN      A      192.168.100.94
```

Service discovery DNS with port

```
$ dig @192.168.100.94 -p 8600 dns.service.consul SRV
```

```
;; ANSWER SECTION:
```

```
dns.service.consul. 0 IN SRV 1 1 8500 agent-three.node.dc1.consul.
```

```
;; ADDITIONAL SECTION:
```

```
agent-three.node.dc1.consul. 0 IN A 192.168.100.94
```

服务注册

- 人工（开发一个平台？）
- 服务生成（第三方注册）
- 服务内部注册（目前的方式）

原则上都须服务拥有者（或知情者）提供服务相关信息，以上皆属提供方式的变化

服务注册--人工

May provide as option for emergency case (shell can already do it)

I have principle that try not have human to intervention(it's error-prone)

How to deal the repetition of deregister and register (it's boring to repeat)

服务注册--第三方注册

服务生成外部信息, 即通过一定方式(输出一个service.json文件/或通过日志)将服务信息暴露出来, 再由第三方注册程序获取后进行分析注册

好处: 服务自身无需实现, 注册功能(虽然只是一个http调用)

坏处: 更难于管理, 第三方注册的位置, 检测变动的频率, 一定的延迟, 注册的频率与检测的频率不符, 并且带来额外的检测

服务注册--服务内部注册 (current)

需要服务进行HTTP请求进行注册(与服务发现HTTP请求类似)

服务信息与服务本身保持同步(更容易维护)

服务拥有者的自由性更高

We often register the service when we start running the service (de-register it when it stop)

服务健康检测

- Script + Interval (any command, often at local)
- **HTTP + Interval** (any url , http get) (recommended)
- TCP + Interval (tcp connection test)
- TTL (service send heartbeat back to consul)
- Docker + Interval

Scripts

- Inline script
- Script file

好处: 可编写复杂的检测(同时维护上也带来一定的难度)

坏处: script file 须以本地文件的方式, 管理上也存在难度

HTTP

通过检测服务暴露的运行时健康状态进行监控(可控制频率)

好处:简单

坏处:需提供http服务用于检测服务状态,如服务本身不包含http功能,则须单独提供

视语言不同(如Go语言提供http服务/health 终端,则非常简单)

TCP

基于是否能建立tcp连接进行测试

适用于未提供未提供http等特殊类型的连接检测

TTL

由服务进行主动的向注册机发送TTL(心跳)消息, 来维持活着的状态, 超过一定时间仍未接收到TTL消息, 则认为检测失败

`/v1/agent/check/pass/<checkId>`

When this endpoint is accessed via a GET, the status of the check is set to passing and the TTL clock is reset


Example of service register

http://127.0.0.1:8500/v1/agent/service/register

The register endpoint expects a JSON request body to be PUT

The **Name** field is mandatory, If an **ID** is not provided, it is set to **Name**

Tags, **Address**, **Port** and **Check** are optional



```
{
  "ID": "redis1",
  "Name": "redis",
  "Tags": [
    "master",
    "v1"
  ],
  "Address": "127.0.0.1",
  "Port": 8000,
  "Check": {
    "Script": "/usr/local/bin/check_redis.py",
    "HTTP": "http://localhost:5000/health",
    "Interval": "10s",
    "TTL": "15s"
  }
}
```

APIs

- [api](#) - Official Go client for the Consul HTTP API
- [consulate](#) - Python client for the Consul HTTP API
- [python-consul](#) - Python client for the Consul HTTP API
- [consul-php-sdk](#) - PHP client for the Consul HTTP API
- [scala-consul](#) - Scala client for the Consul HTTP API
- [consul-client](#) - Java client for the Consul HTTP API
- [consul-api](#) - Java client for the Consul HTTP API
- (and more: Erlang, Ruby, Node.js, C#)

Tools

- [Envconsul](#) - Read and set environmental variables for processes from Consul.
- [Consul Replicate](#) - Consul cross-DC KV replication daemon.
- [Consul Template](#) - Generic template rendering and notifications with Consul
- [Consul Migrate](#) - Data migration tool to handle Consul upgrades to 0.5.1+
- [consul-cli](#) - Command line interface to Consul HTTP API

Consul-template

Generic template rendering and notifications with Consul

Queries a Consul instance and updates any number of specified templates on the file system

Can optionally run arbitrary commands when the update process completes

Example

```
$ consul-template \  
-consul 127.0.0.1:8500 \  
-template "/tmp/template.ctmpl:/var/www/nginx.conf:service nginx restart" \  
-retry 30s \  
-once
```

Templating Language

- API Functions

File, key, service, node, ls, tree, etc ...

- Helper Functions

byKey, byTag, contains, in, loop, join, trimSpace, parseJSON, plugin, regexMatch, split, timestamp, toJSON, toLower, toTitle, toUpper, toYAML, etc...

- Math Functions

Add, subtract, multiply, divide

HAProxy-config example (haproxy.ctmpl)

```
global
  daemon
  maxconn {{key "service/haproxy/maxconn"}}
defaults
  mode {{key "service/haproxy/mode"}}{{range ls "service/haproxy/timeouts"}}
  timeout {{.Key}} {{.Value}}{{end}}
listen http-in
  bind *:8000{{range service "release.web"}}
  server {{.Node}} {{.Address}}:{{.Port}}{{end}}
```

Run consul-template

```
$ consul-template \  
-consul 127.0.0.1:8500 \  
-template haproxy.ctmpl:/etc/haproxy/haproxy.conf  
-dry
```

**-dry means Dump generated templates to the console. If specified, generated templates are not committed to disk and commands are not invoked*

Example outputs

global

daemon

maxconn 4

defaults

mode default

timeout 5

listen http-in

bind *:8000

server nyc3-worker-2 104.131.109.224:80

server nyc3-worker-3 104.131.59.59:80

server nyc3-worker-1 104.131.86.92:80

Envconsul

Read and set environmental variables for processes from Consul

```
$ envconsul \  
-consul=demo.consul.io \  
-prefix redis/config \  
-once \  
env  
ADDRESS=1.2.3.4  
PORT=55
```

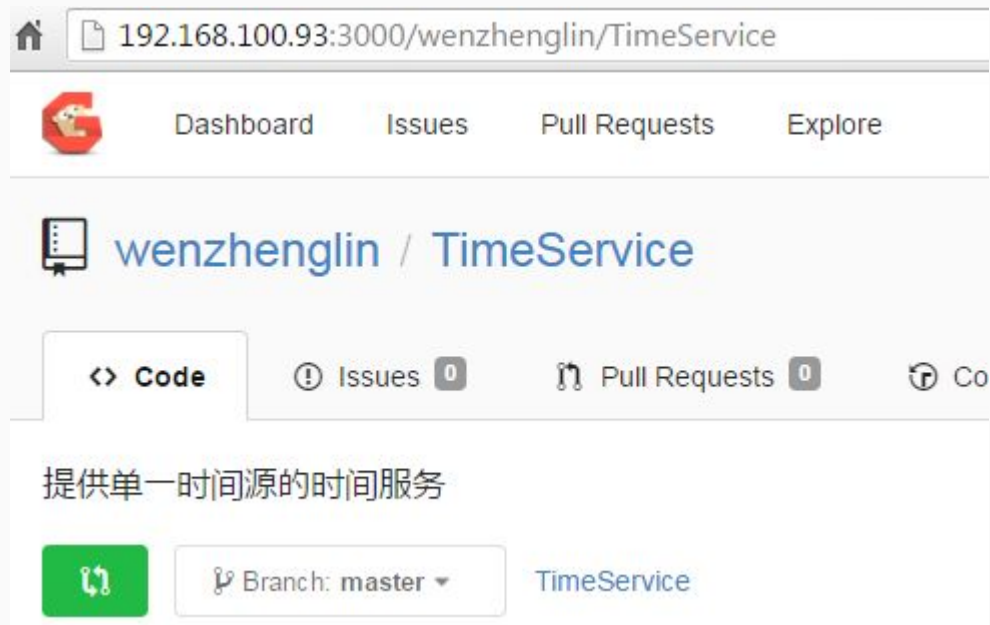
Consul-replicate

Consul cross-DC KV replication daemon

This makes it possible to manage application configuration from a central, with low-latency asynchronous replication to other data centers

Avoiding the need for smart clients which would need to write to all data centers and queue writes to handle network failures

Service discovery example



The screenshot shows the GitHub interface for the repository `wenzhenglin / TimeService`. The address bar displays `192.168.100.93:3000/wenzhenglin/TimeService`. The repository description is "提供单一时间源的时间服务". The main branch is `master`. The repository has 0 issues and 0 pull requests.

192.168.100.93:3000/wenzhenglin/TimeService

Dashboard Issues Pull Requests Explore

wenzhenglin / TimeService

<> Code ⓘ Issues 0 🔄 Pull Requests 0 🏷 Co

提供单一时间源的时间服务

🔄 Branch: master TimeService

运行

```
cd bin
./timeServer
```

服务访问

可采用grpc支持的任一语言编写 以下通过go语言实现的客户端进行示例

```
cd bin
./timeClient
```

示例输出

```
# ./timeClient
Time is: 2016-05-05 14:14:45.241
#
```

TimeService register example

```
check := consul.AgentServiceCheck{
    TCP:    addr + ":" + port,
    Interval: "10s",
    Timeout: "3s",
}
```

```
service := consul.AgentServiceRegistration{
    Name: "TimeService",
    Tags: []string{"master"},
    Port: p,
    Address: addr,
    Check: &check,
}
```

```
conf := consul.DefaultConfig()
if consuladdr != "" {
    if consulport == "" {
        consulport = "8500"
    }
    conf.Address = consuladdr + ":" + consulport
}

client, err := consul.NewClient(conf)
if err != nil {
    return fmt.Errorf("create client error: %v\n", err)
}

return client.Agent().ServiceRegister(&service)
```

TimeService discovery example

```
func discover() (string, error) {  
    client, err := consul.NewClient(consul.DefaultConfig())  
    if err != nil {  
        return "", fmt.Errorf("create client error: %v\n", err)  
    }  
  
    services, _, err := client.Catalog().Service(  
        "TimeService",  
        "master",  
        &consul.QueryOptions{},  
    )  
}
```

Fileserver2 shell version register

```
service="$( cat <<eof
{
  "Name": "fileserver2",
  "Tags": [
    "master"
  ],
  "Address": "192.168.100.94",
  "Port": 8000,
  "Check": {
    "HTTP": "http://localhost:8000",
    "Interval": "3s"
  }
}
eof
)"

url="http://localhost:8500/v1/agent/service/register"
curl -X PUT "$url" -d "$service"
```



```
-# de-register a service
```

```
service="fileserver2"
```

```
url="http://localhost:8500/v1/agent/service/deregister/$service"
```

```
curl "$url"
```

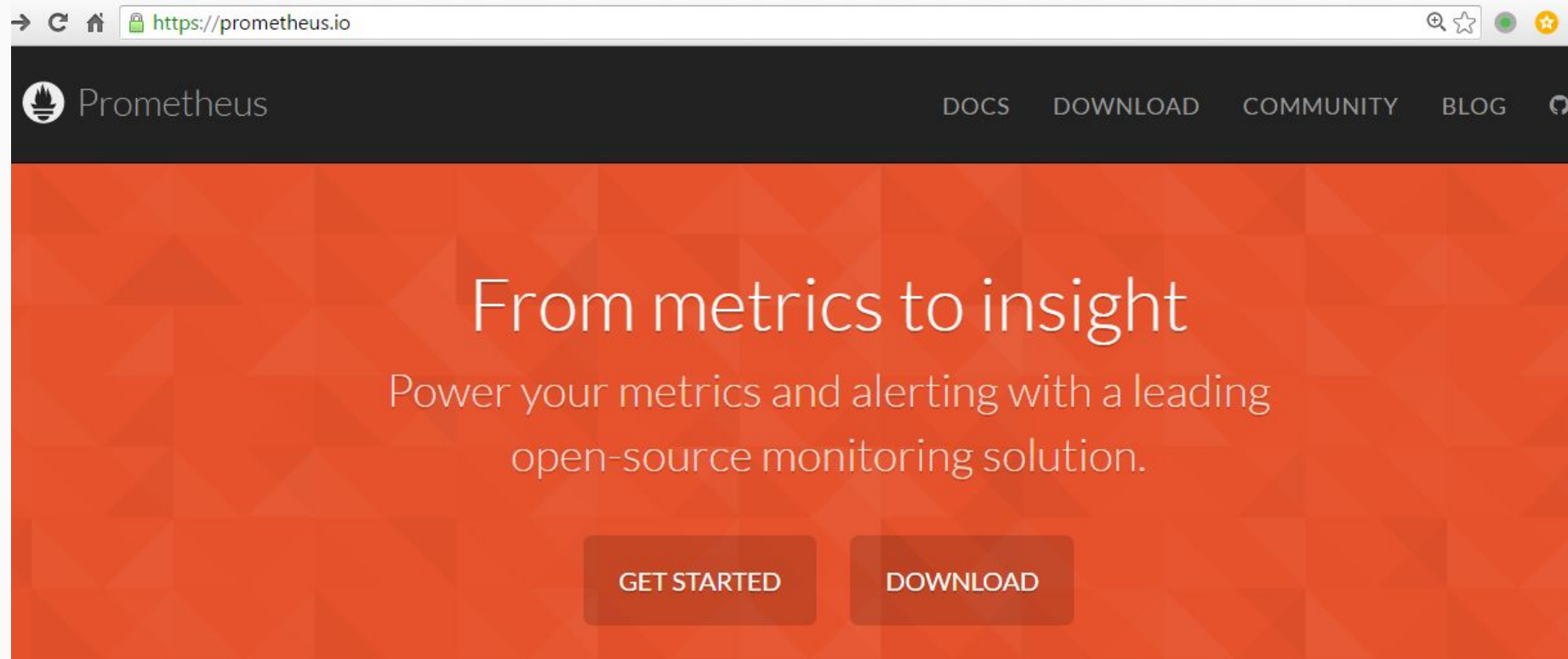
About monitoring

Service advance monitoring

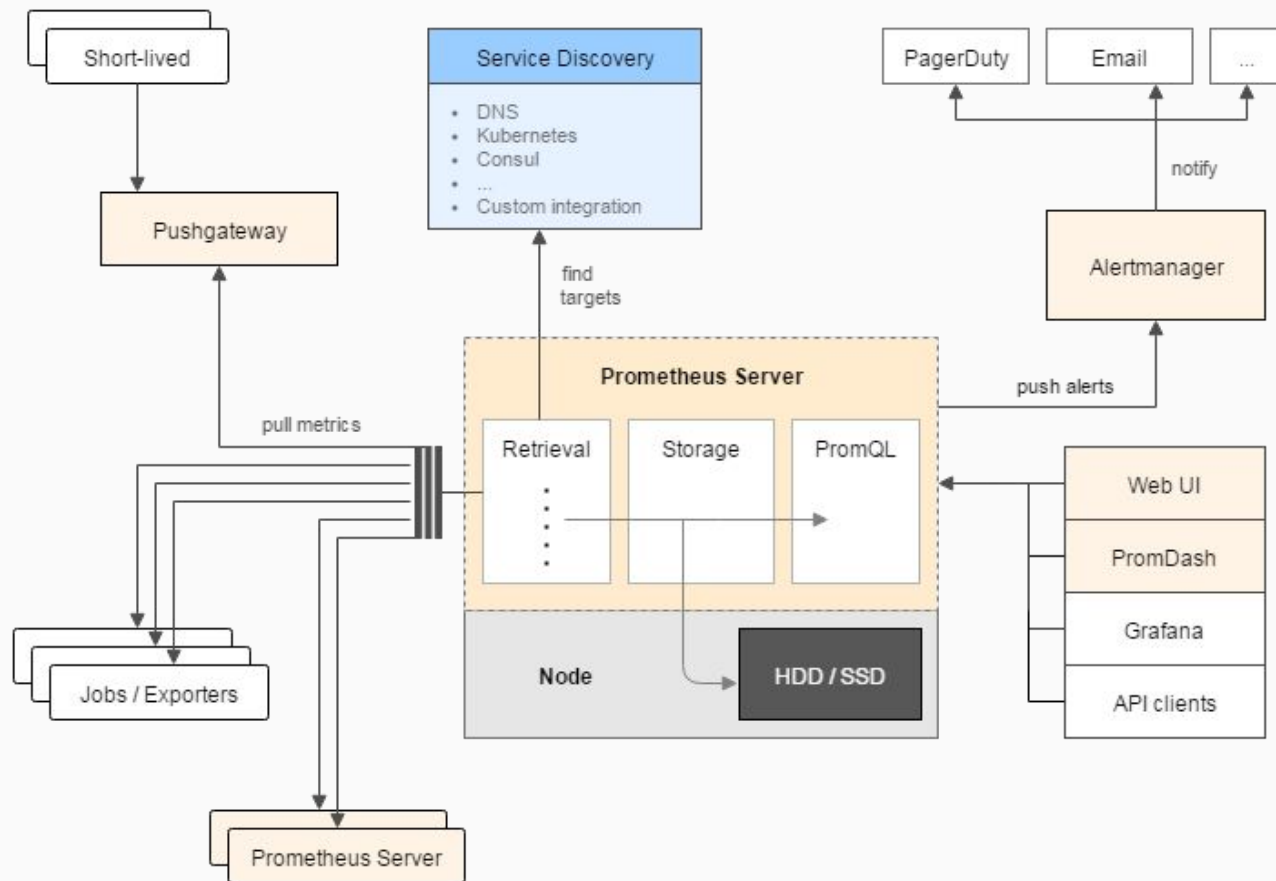
Still work-in-progress

A independent monitoring platform (more advance)

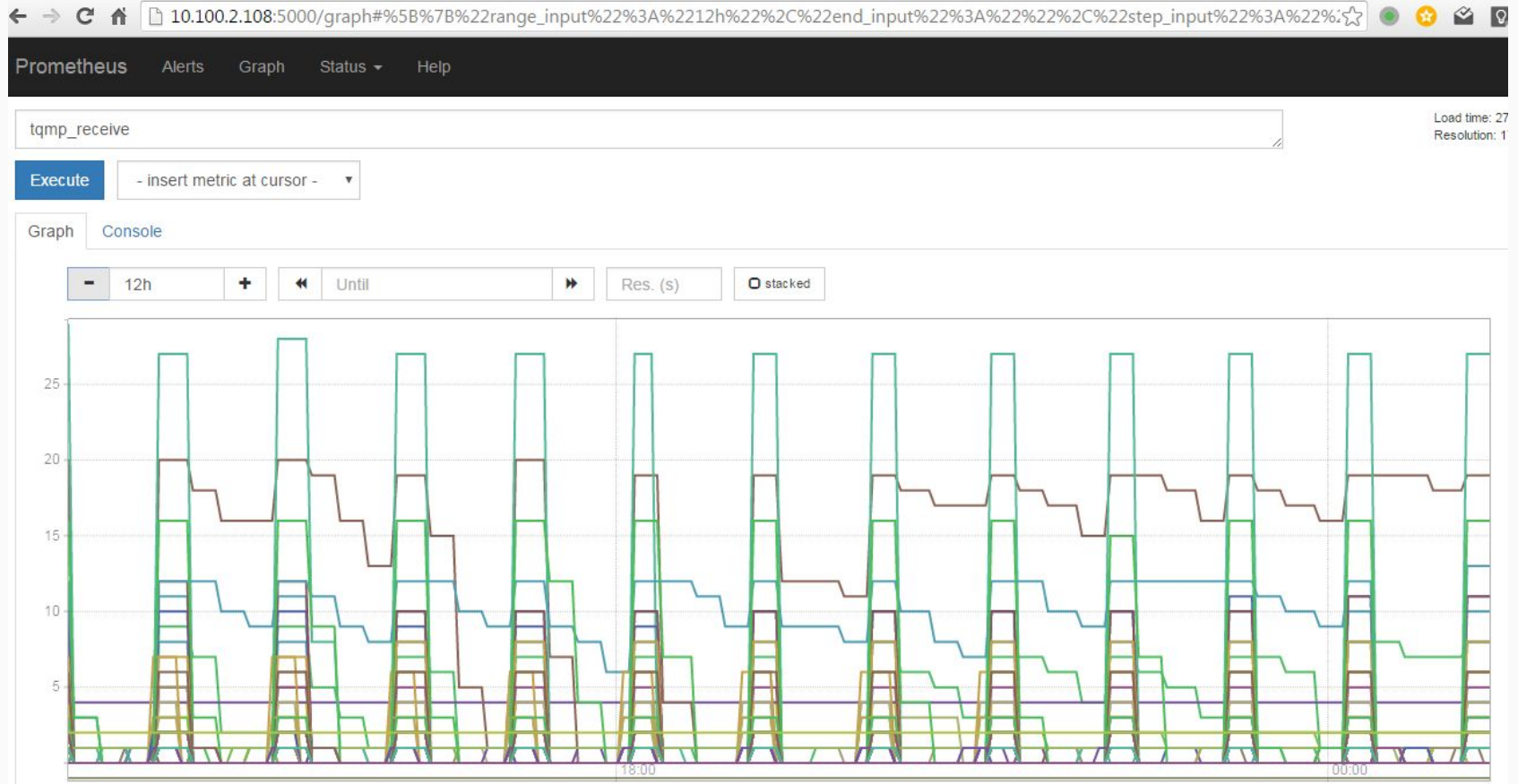
May primary monitor through metrics, log aggregate and analysis or code instrumentation



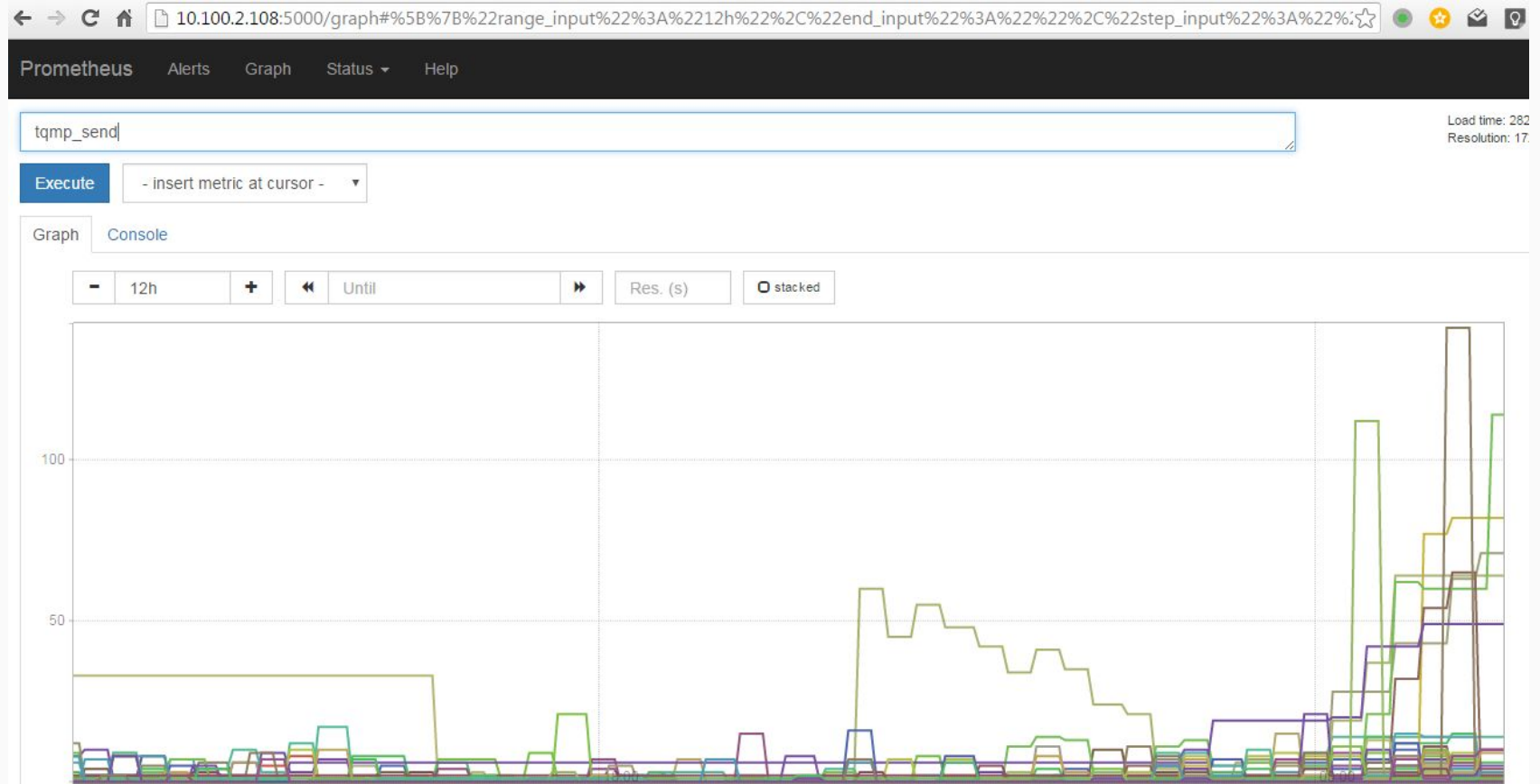
Prometheus



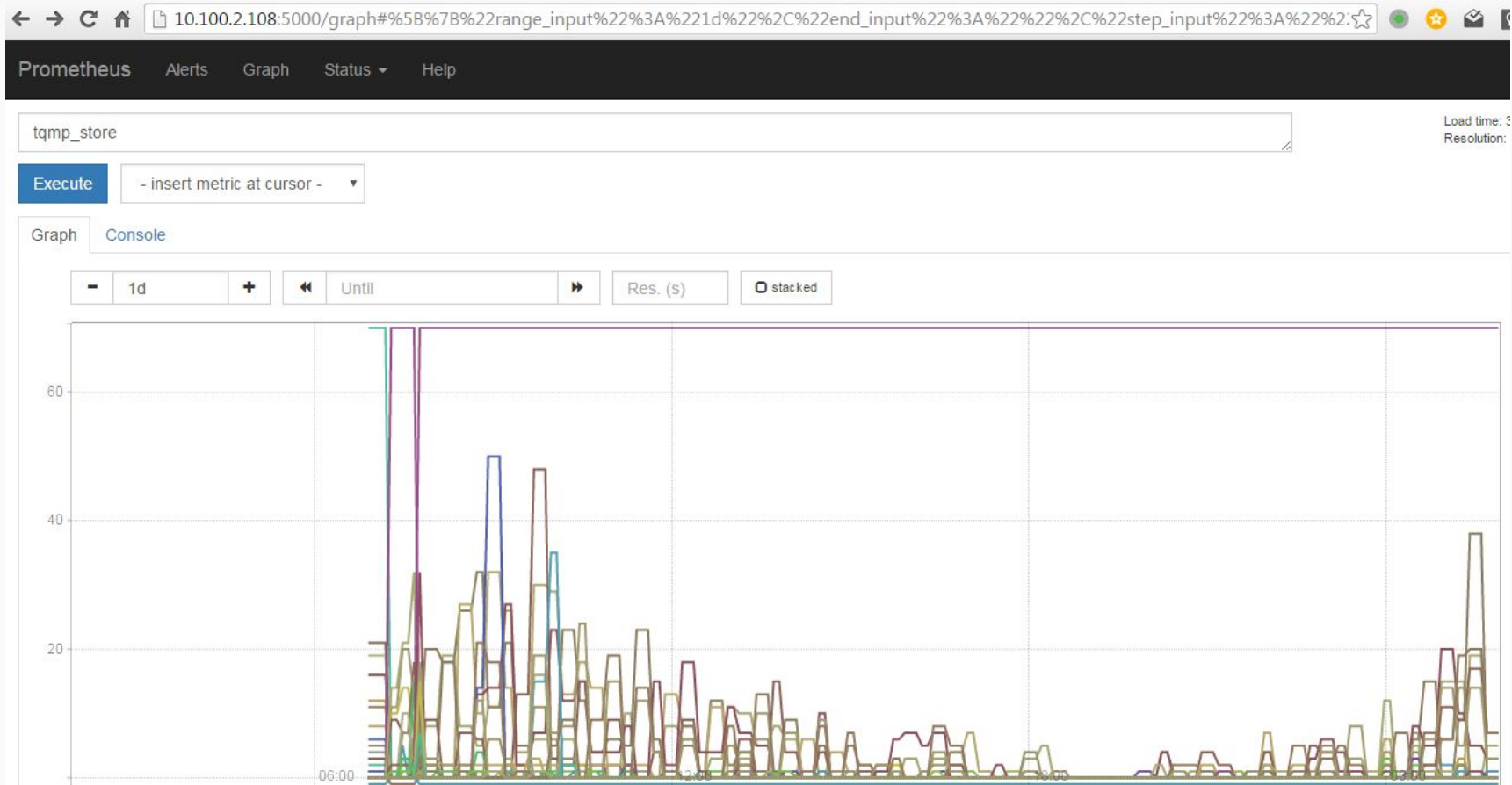
Prometheus example1



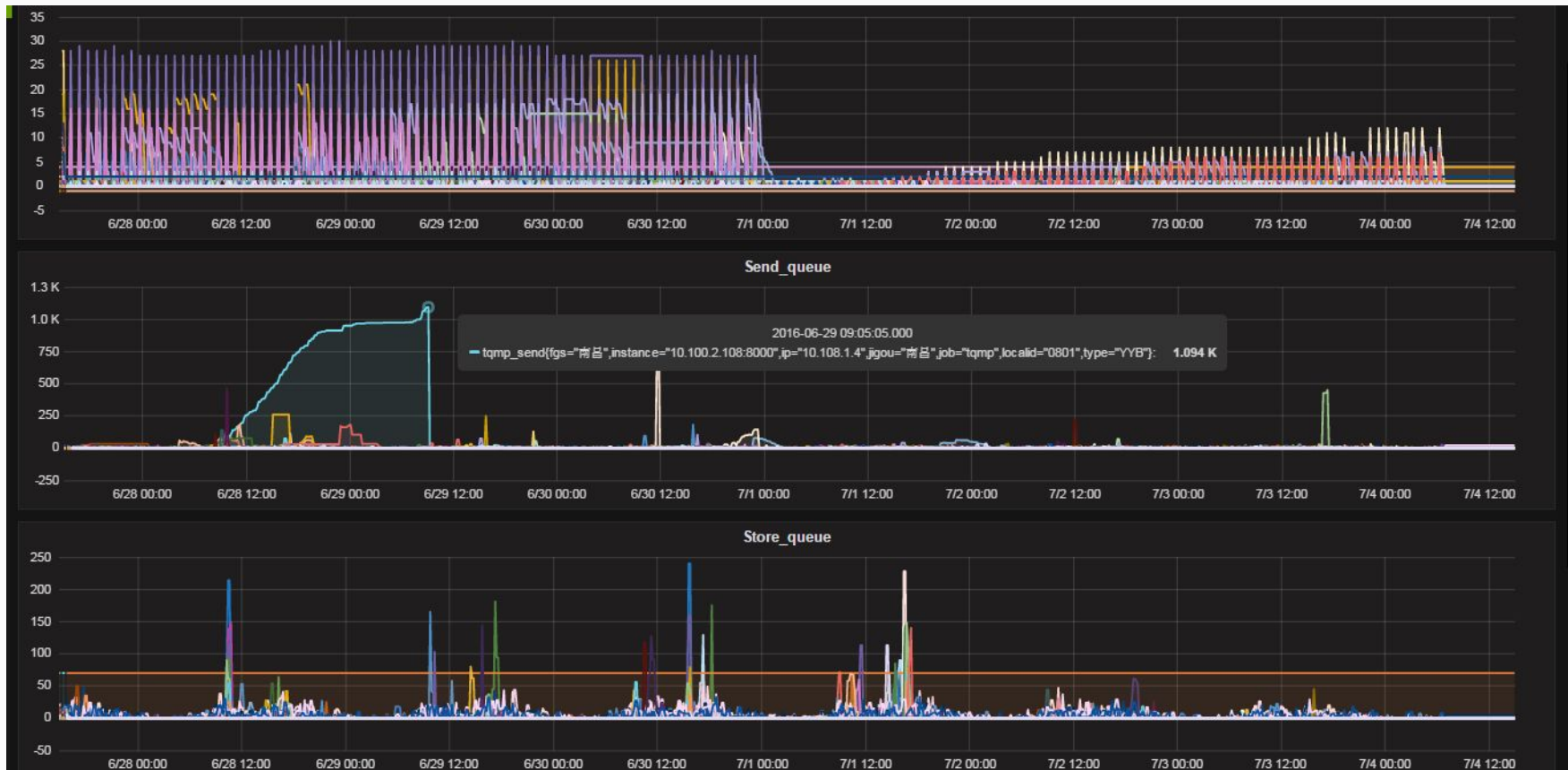
Prometheus example2



Prometheus example3



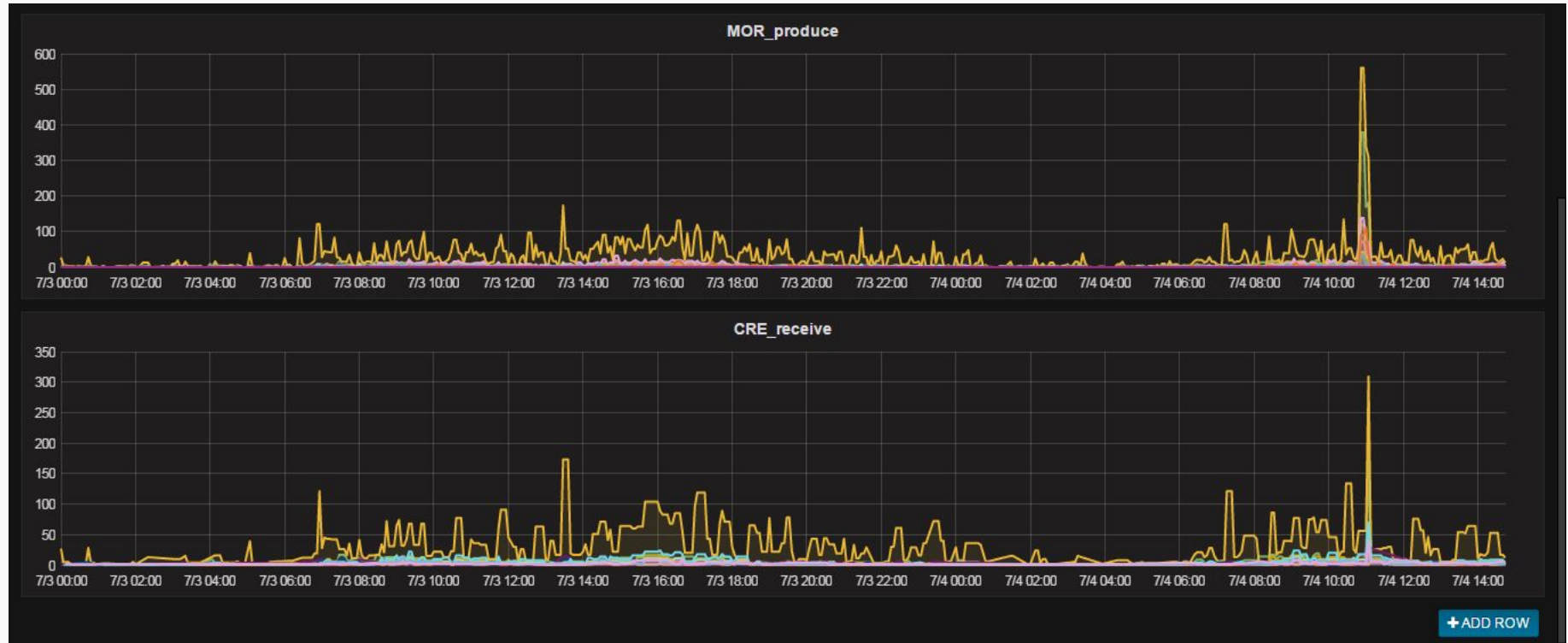
Grafana example -- TQMP



Grafana example -- xbcenterExchange



Grafana example -- xbcenterExchange (cont.)



Some thought

Don't make it complicated

Not recommend to use the following feature

- Coordinate
- Leader election
- Lock

Suggest to use the basic functions only (service discovery), Complicated will weak you program, make it hard to diagnose and fix and recover

Need a person to manage it all?

Say service register, create checks, etc

We tend to let developer as free as possible

Not a must, but will provide consultant support

Summary

Summary

Service Discovery introduction

Comparison

Introduce Consul

Service discovery example

Some thought

Summary (cont.)

We now have a solution for service discovery with basic monitoring

Later we will try to find out a way for advance monitoring

As service discovery and service monitoring together toward a better service management solution

Thank you

Wen Zhenglin

wenzhenglin@bjjdsy.com.cn

2016-6-28

