



Implementing distributed systems with



Matúš Valo

Distributed systems

- Programming distributed systems is **hard**
- Challenges [1]:
 - Service access and configuration
 - Event handling
 - Concurrency
 - Synchronization
- Single point of failure problem
- Split brain problem
- CAP theorem



Failover Celery Beat

- **celery beat** is a scheduler; It kicks off tasks at regular intervals, that are then executed by available worker nodes in the cluster.

The screenshot shows the GitHub interface for the 'celery/celery' repository. At the top, the repository name 'celery / celery' is displayed. To the right, there are buttons for 'Watch' (398), 'Star' (8,824), and 'Fork' (2,521). Below this, a navigation bar includes links for 'Code', 'Issues' (273), 'Pull requests' (38), 'Projects' (0), 'Wiki', and 'Insights'. The 'Issues' tab is selected. The main content area shows an issue titled 'Failover for celerybeat #1495'. To the right of the title is a green button labeled 'New issue'. Below the title, there is a green badge with an exclamation mark and the word 'Open', followed by the text 'kerwin opened this issue on Aug 1, 2013 · 23 comments'.



Highly Available Airflow

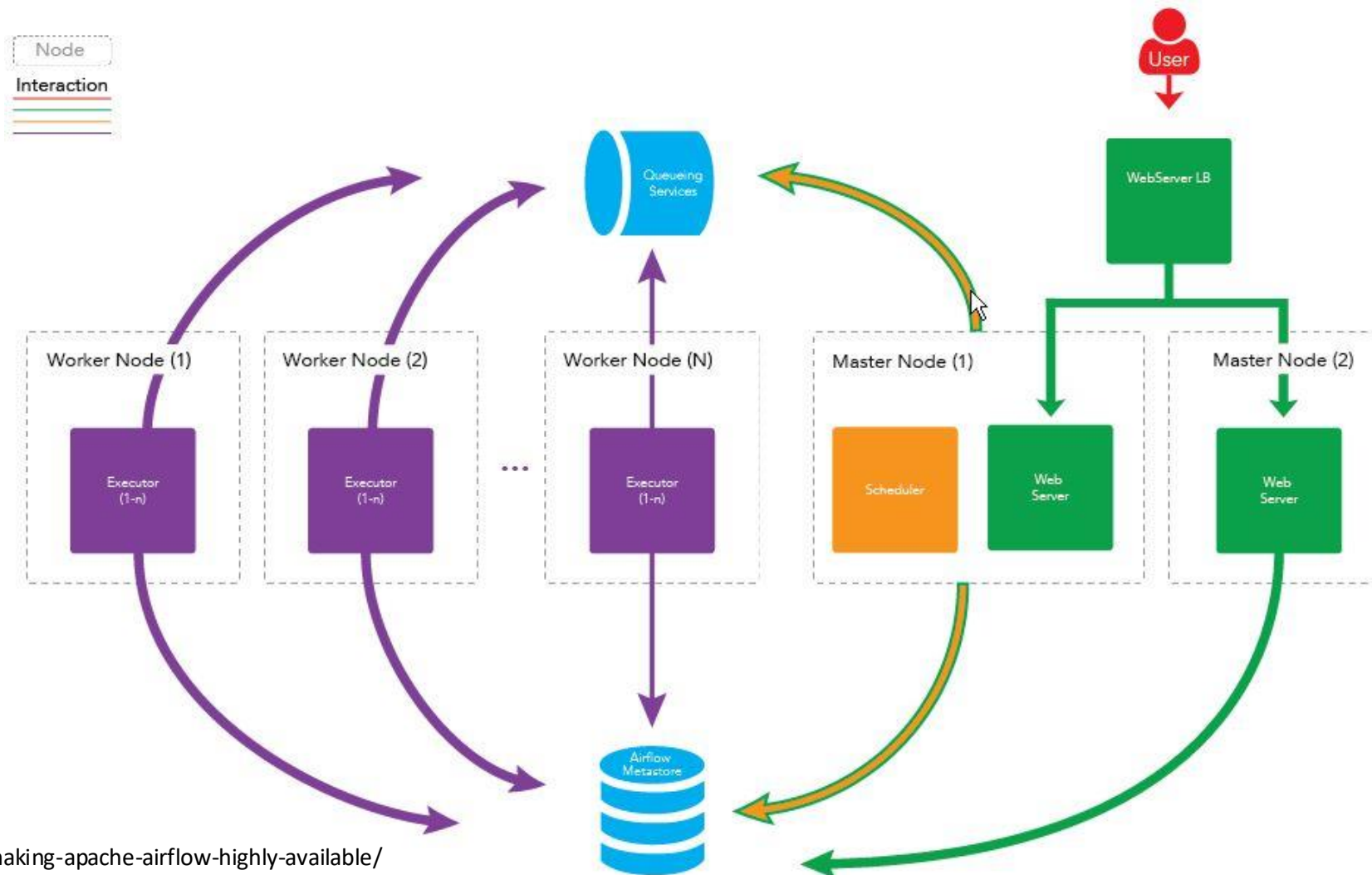
- Airflow is a platform to programmatically author, schedule and monitor workflows.
- Workflows are Directed acyclic graphs (DAGs) of tasks
- Scheduler executes tasks on an array of workers while following the specified dependencies.
- Currently, Airflow has single point of failure – Scheduler [1]
- Multiple attempts exists for solving this issue including home-brew solutions [2]

[1] <http://site.clairvoyantsoft.com/making-apache-airflow-highly-available/>

[2] <https://github.com/teamclairvoyant/airflow-scheduler-failover-controller>



Highly Available Airflow





Why Consul?

- **Consul is clustered and highly available.**
- **Consul enables easy development of distributed and highly available systems.**
- Consul makes it simple for services to register themselves and to discover these services via a DNS or HTTP interface.
- Pairing service discovery with health checking prevents routing requests to unhealthy hosts and enables services to easily provide circuit breakers.
- Consul scales to multiple datacenters out of the box with no complicated configuration. Look up services in other datacenters, or keep the request local.
- Flexible key/value store for dynamic configuration, feature flagging, coordination, leader election and more.



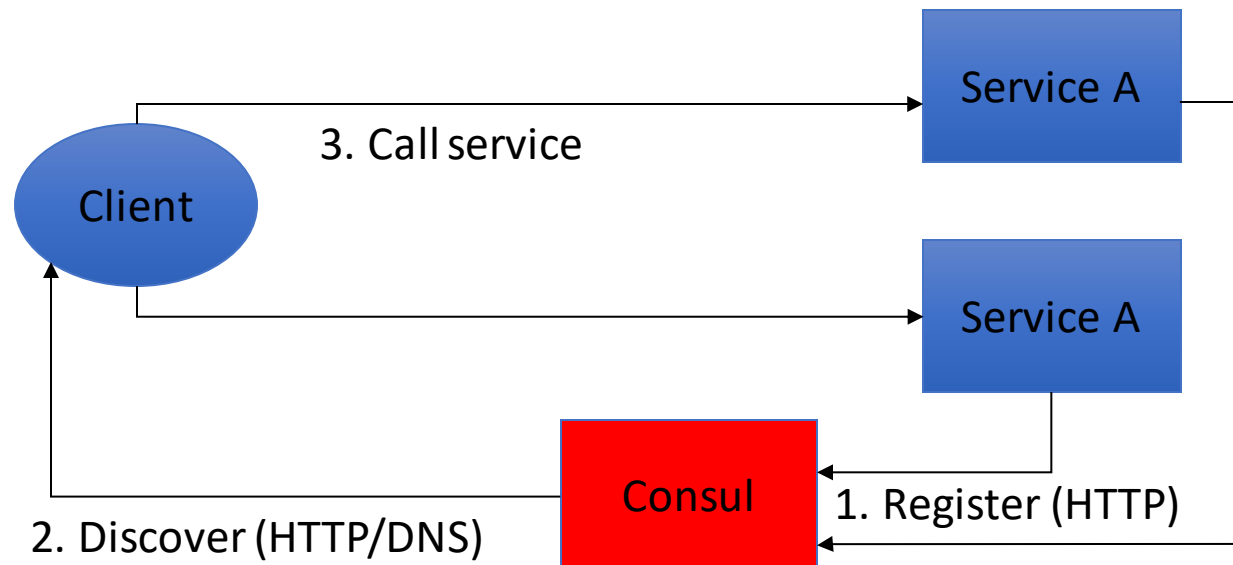
Consul Cluster

- Consul cluster consists from at least 3 nodes (supports 1 node failure)
- When higher failure tolerance is needed:
 - 5 nodes (supports 2 node failure)
 - 7 nodes (supports 3 node failure)
- When new cluster is created it needs to be bootstrapped:
 - each node is started in bootstrap mode using `-bootstrap-expect BNODES` parameter. BNODES is the initial number of nodes in cluster.
 - nodes are joined together using `consul join` command
- consul supports special single node DEV mode for developing purposes.



Service Discovery

- Useful for scalable distributed system, where new nodes needs to be discovered automatically.
- Distributed component can register at startup and from that point is visible



- Service Registration

```
$ cat servicea.json
```

```
{  
  "ID": "servicea1",  
  "Name": "serviceA",  
  "Tags": ["primary", "v1"],  
  "Address": "servicea.example.com",  
  "Port": 8000  
}
```

```
$ curl -X PUT -d @servicea.json localhost:8500/v1/agent/service/register
```

- Service discovery (DNS Interface)

```
$ dig -p 8600 servicea.service.consul @consul.example.com
```

```
/// OUTPUT OMITTED
```

```
;; ANSWER SECTION:
```

```
servicea.service.consul. 0      IN      CNAME   servicea.example.com.
```

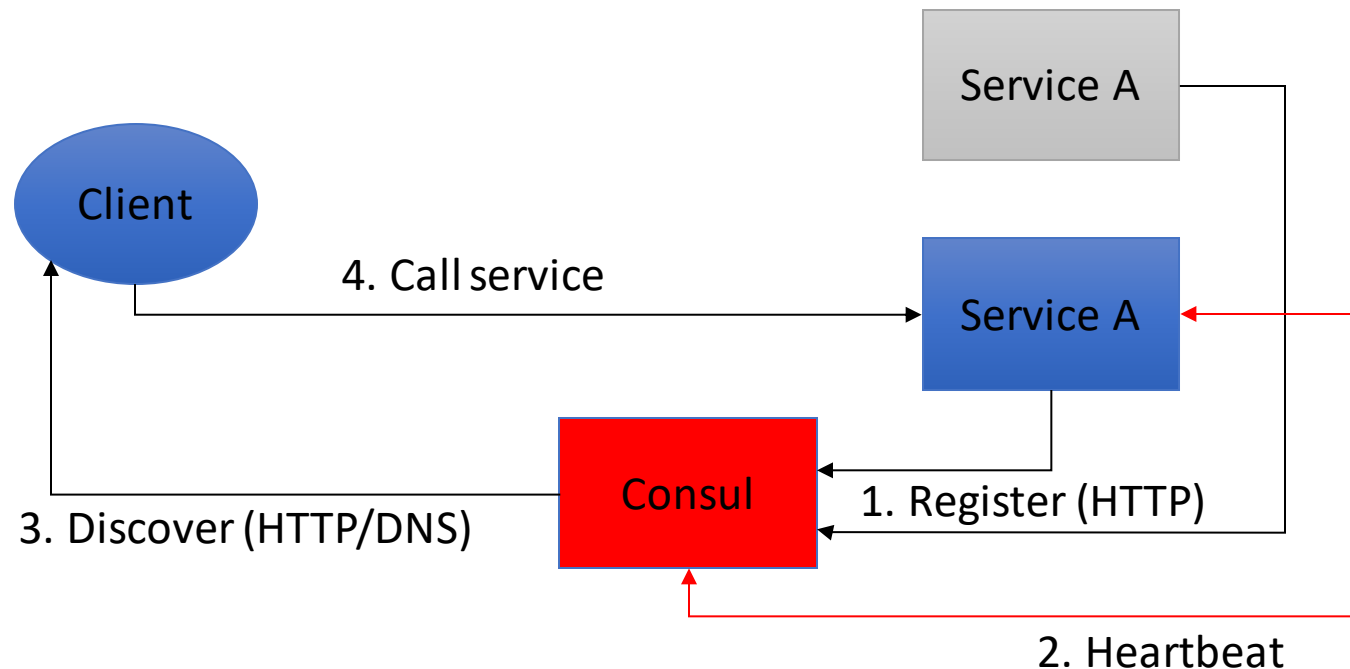
```
/// OUTPUT OMITTED
```

```
$ dig -p 8600 servicea.service.consul @consul.example.com SRV
```



Health check

- Service can ask consul to track availability of service using heartbeat (HB)
- Multiple ways of HB are supported: HTTP, raw TCP/IP, Custom Script, Docker container check



- Service registration

```
$ cat servicea.json
```

```
{  
  "ID": "servicea1",  
  "Name": "serviceA",  
  "Tags": ["primary", "v1"],  
  "Address": "servicea.example.com",  
  "Port": 8000,  
  "Check": {  
    "DeregisterCriticalServiceAfter": "90m",  
    "HTTP": "http://servicea.example.com:5000/health",  
    "Interval": "10s"  
  }  
}
```

```
$ curl -X PUT -d @servicea.json localhost:8500/v1/agent/service/register
```

- Service discovery (HTTP interface)

```
$ curl localhost:8500/v1/health/service/servicea?passing=true
```

```
[
  {
    "Node": {
      "ID": "3114d0f7-ff8b-fcc9-a8b8-82fde8cc6a4f",
      "Node": "mynode.example.com",
      "Address": "192.168.10.10",
      "Datacenter": "dc1",
      /// OUTPUT OMMITED
    },
    "Service": {
      "ID": "servicea1",
      "Service": "serviceA",
      "Tags": ["primary", "v1"],
      "Address": "servicea.example.com",
      "Port": 8000,
      /// OUTPUT OMMITED
    },
    "Checks": [ /// OUTPUT OMMITED ]
  }
]
```



K/V Storage

- Consul provides Key/Value Storage which is distributed and highly available.
- Any data can be stored in form “key”: “value”
 - “value” can store any JSON value encoded as base64
- K/V Storage has CLI or HTTP API (easy to use)
- Consul also provides additional functionality:
 - transactions – multiple operations on K/V storage with atomic execution.
 - atomic key updates using a Check-And-Set operation

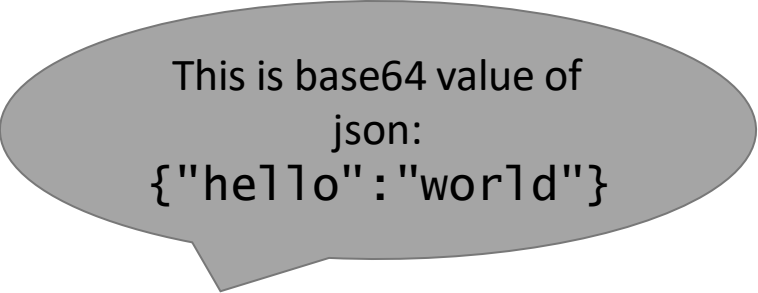
- Setting value to K/V storage

```
$ curl -X PUT -d '{"hello":"world"}' localhost:8500/v1/kv/hello
true
```

- Getting value from K/V storage

```
$ curl localhost:8500/v1/kv/hello
[
```

```
  {
    "LockIndex": 0,
    "Key": "hello",
    "Flags": 0,
    "Value": "eyJ0ZWxsbyI6Indvcmxkin0=",
    "CreateIndex": 87,
    "ModifyIndex": 87
  }
```



This is base64 value of
json:
{"hello":"world"}

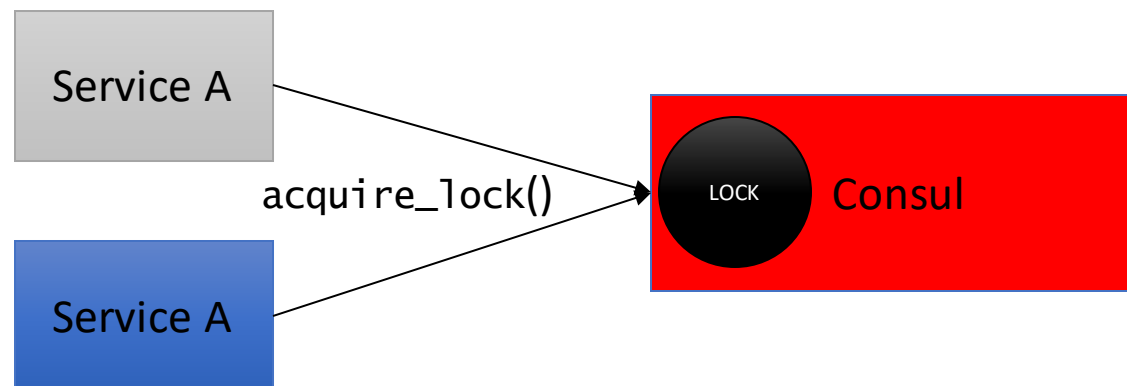
```
]
```

- Deleting value from K/V storage

```
$ curl -X DELETE localhost:8500/v1/kv/hello
true
```

Distributed locks

- Semantic of Consul locks is similar to the mutex (mutual exclusion) but can be used also as a semaphore
- Consul agent must be present on client
- To start process holding lock: `consul lock [options] <LOCK> <CMD>`
- This guarantee that only one single <CMD> is executed in cluster
- When <CMD> exists/crash the lock is automatically freed.



[Unit]

Description=Failover Service

After=consul.service

Requires=consul.service

[Service]

ExecStart=/usr/local/bin/consul lock -verbose <LOCK_NAME> <EXECUTABLE>

Restart=always

KillSignal=SIGQUIT

Type=simple

StandardError=syslog

NotifyAccess=all

[Install]

WantedBy=multi-user.target



Implementing failover Celery Beat

1. Implement custom scheduler with central storage
e.g. DatabaseScheduler from django-celery-beat
2. Start celery beat on at least two nodes using consul distributed lock with custom scheduler

```
$ consul lock celery_beat_lock celery -A proj beat --scheduler  
django_celery_beat.schedulers:DatabaseScheduler
```



Additional Consul features

- Access Control List (ACL) system
- Multi Datacenter
- Commercial support
- Watches and Events
- Web UI
- Libraries for multiple languages (python-consul)



Thank you