

facebook

Command Execution in Heterogeneous Network at Facebook Scale

Surinder Singh

Software Engineer

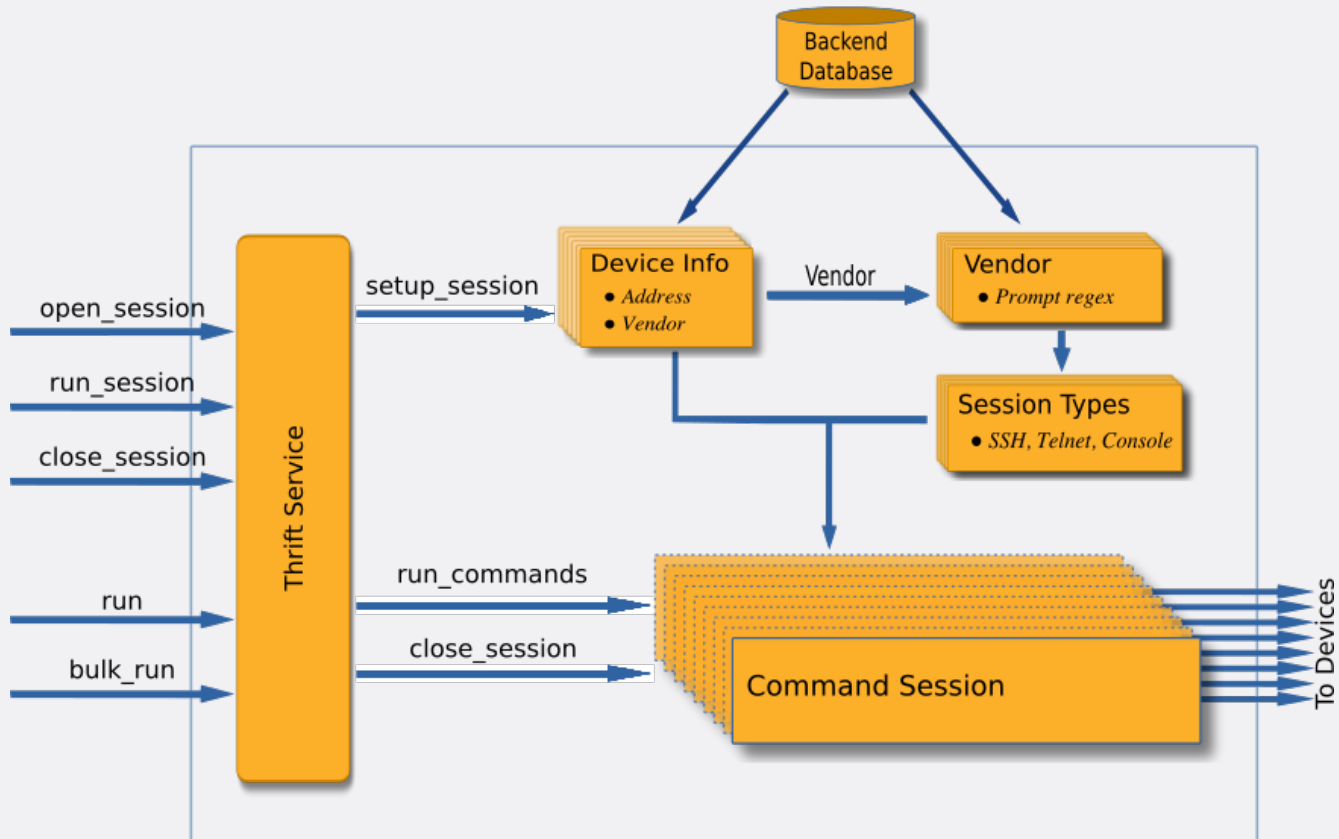
Agenda

- Command Execution on Devices
- FBNet Command Runner
 - APIs
 - Using APIs to run commands on devices
 - Setup FCR service

Command Execution on Devices

- Expect scripts
- Difficult to support multiple vendors
- Difficult to scale to large number of devices
- Support Multiple session types
 - SSH, Console, Thrift, EAPI

FCR: FNet Command Runner



FCR: FBNet Command Runner

- Scalable Thrift service
- Focus on business logic
- Run commands on devices
 - Collect network state
- Multiple language support

FCR: FBNet Command Runner

FCR: FBNet Command Runner

- Python3
 - Asyncio
 - Asyncssh
- Open source
 - <https://github.com/facebookincubator/FCR>

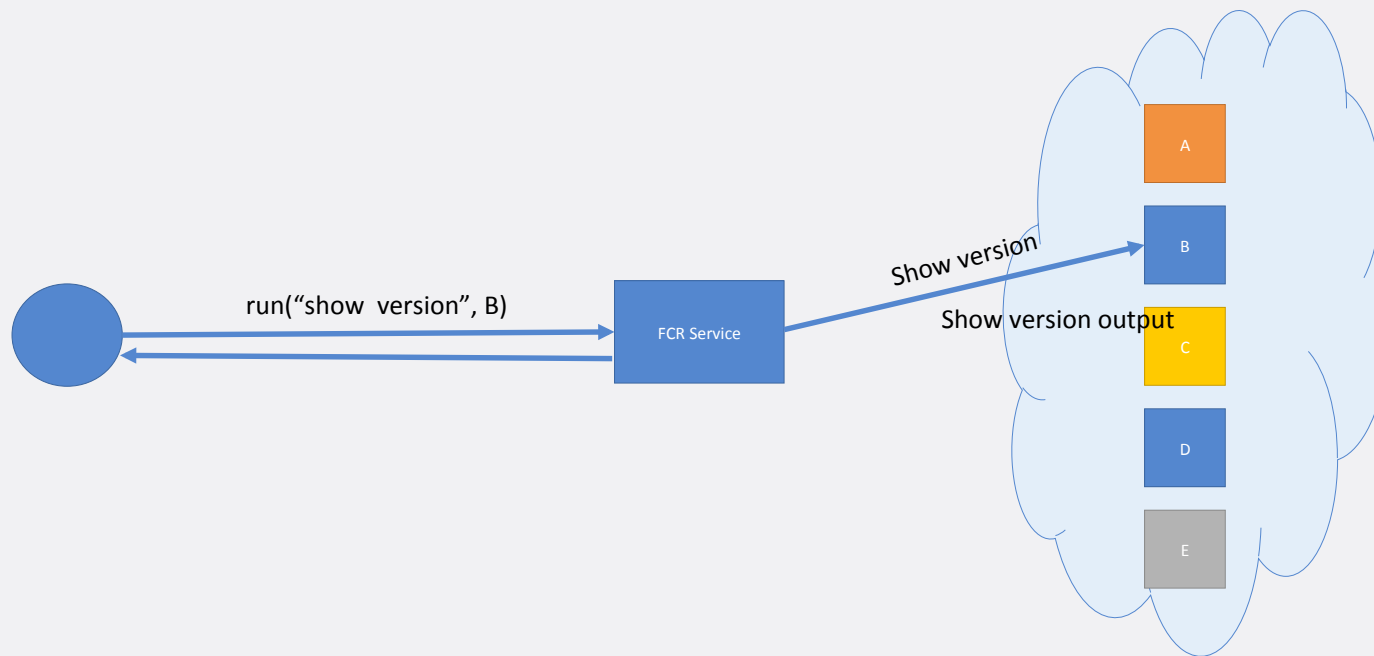
APIs

Running command on a single device

- `CommandResult run(command, Device)`

APIs

Running command on a single device



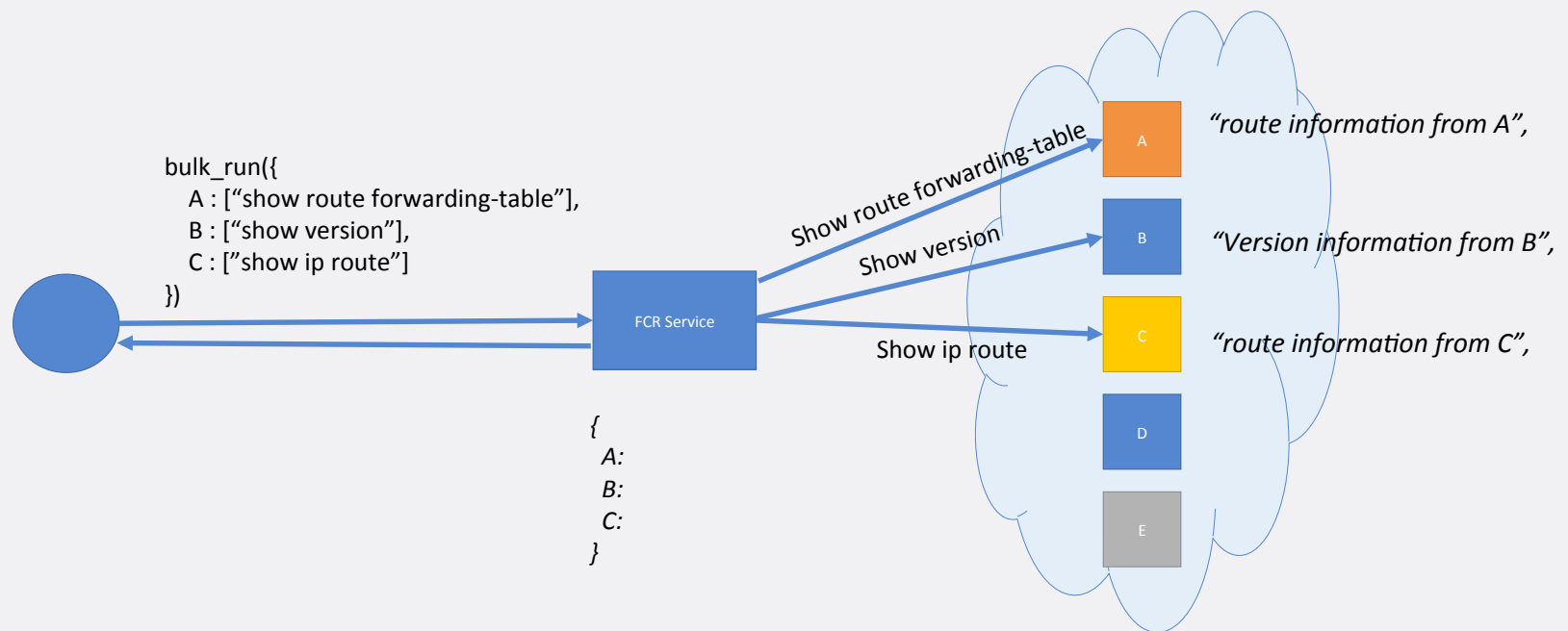
APIs

Running commands in multiple devices in parallel

- Map<device, List<CommandResult>
 bulk_run(device_to_command_map)

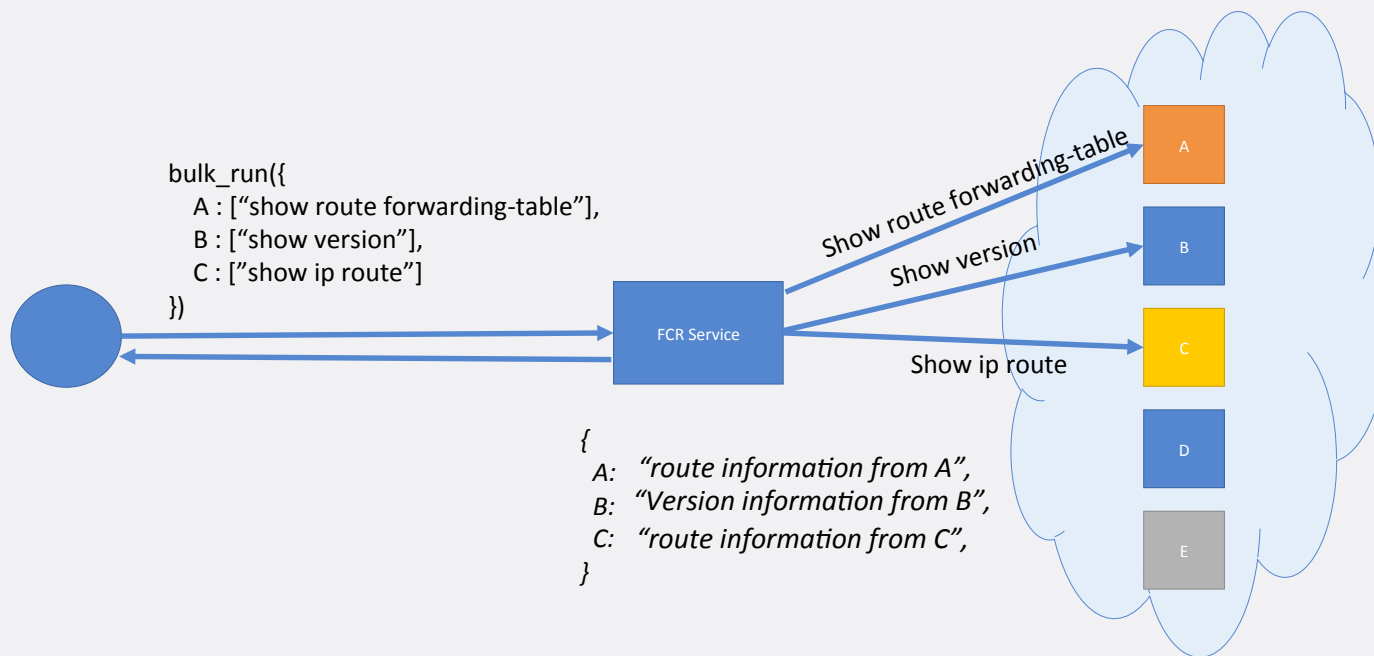
APIs

Running commands on multiple devices



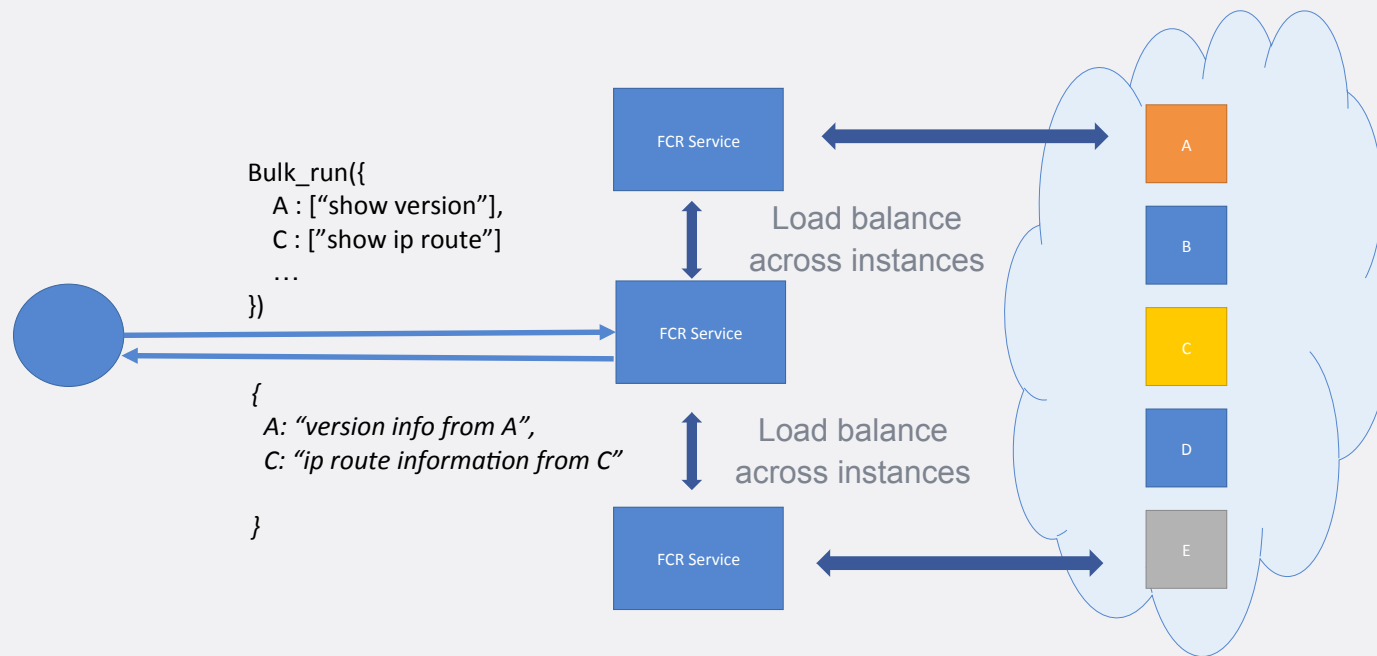
APIs

Running commands on multiple devices



APIs

Running command on multiple devices



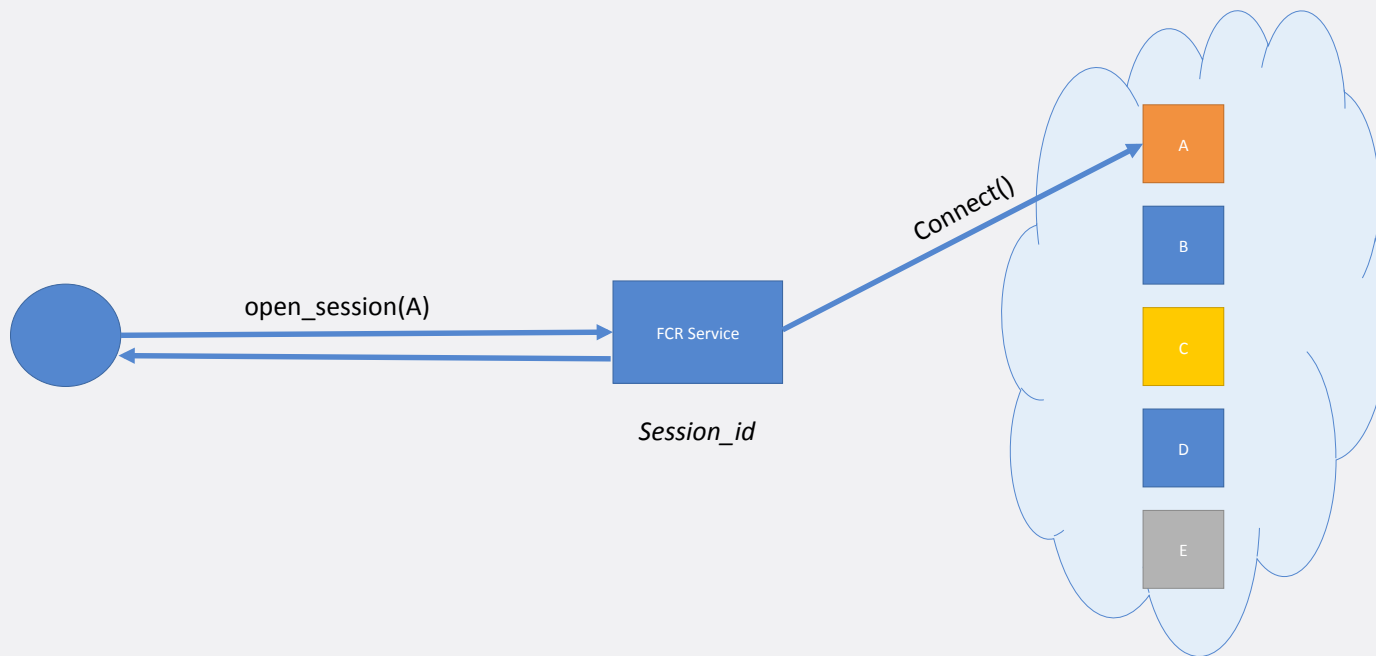
APIs

Interactive APIs

- Session **open_session**(Device)
- CommandResult **run_session**(session, command)
- void **Close_session**(session)

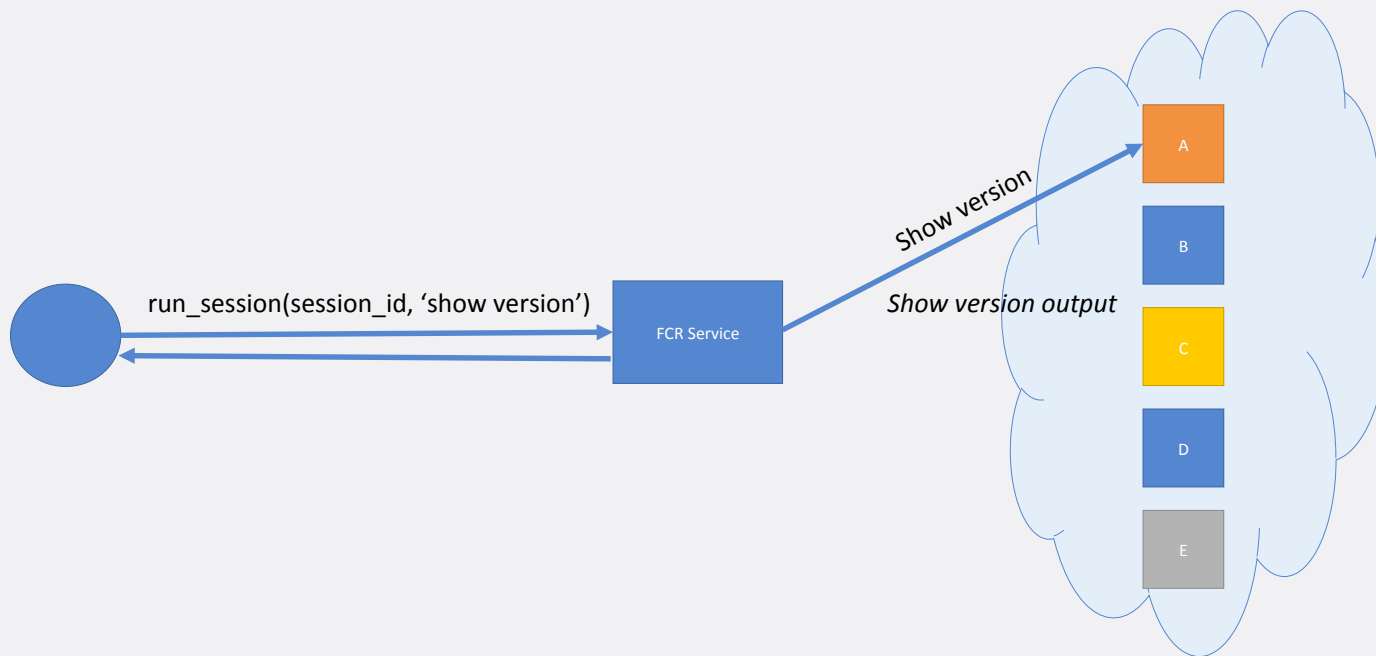
APIs

Interactive APIs



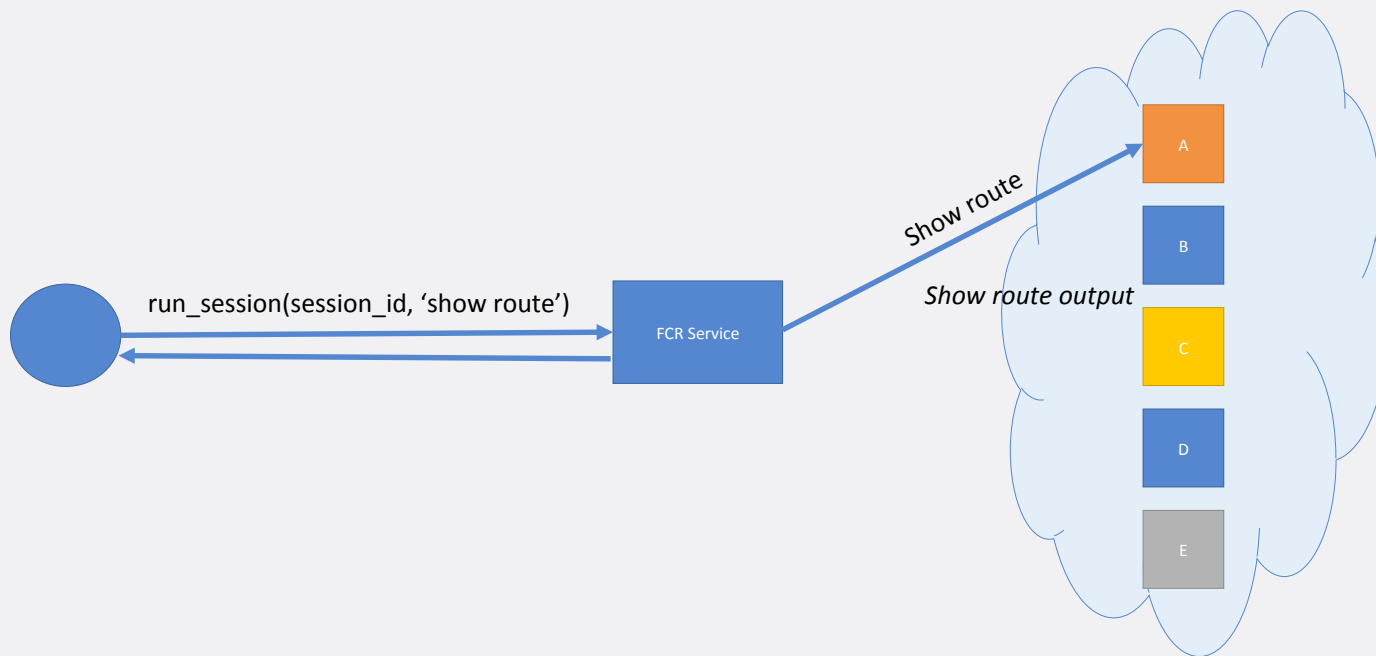
APIs

Interactive APIs



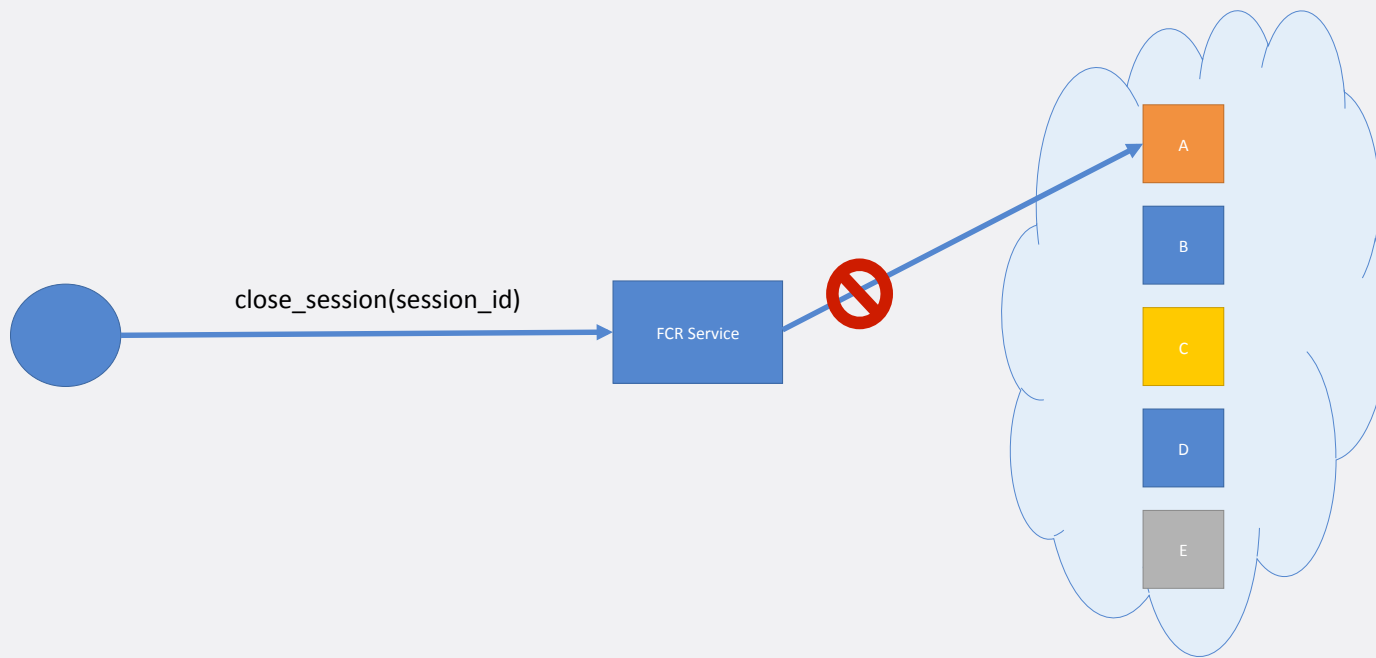
APIs

Interactive APIs



APIs

Interactive APIs



Installation

```
$ git clone --recursive \  
    https://github.com/facebookincubator/FCR.git
```

```
$ python3 -m venv venv
```

```
$ . venv/bin/activate
```

```
$ cd FCR
```

```
$ pip3 install -r requirements.txt
```

```
$ pip3 install .
```

Running Commands

```
def get_client():
```

```
    return AsyncioThriftClient(FcrClient, "localhost", 5000)
```

```
def fcr_device(devname):
```

```
    return fcr_ttypes.Device(devname, username="netbot", password="bot1234")
```

```
async def run_one(cmd, devname):
```

```
    async with get_client() as client:
```

```
        res = await client.run(cmd, fcr_device(devname))
```

```
        print_results(devname, [res])
```

Running Commands (Bulk)

```
async def demo():  
    dev_to_cmd = {}  
  
    for d in ['agg.c1', 'agg.c2']:  
        dev_to_cmd[fcf_device(d)] = ['show version', 'show ip route 11.1.1.3']
```

```
async with get_client() as client:  
    res = await client.bulk_run(dev_to_cmd)
```

```
for devname, results in res.items():  
    print_results(devname, results)
```

Running Commands (Bulk)

```
=====agg.c2=====
=====success=====
```

```
agg.c2# show version
Quagga 0.99.23.1 ().
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

```
=====success=====
```

```
agg.c2# show ip route 11.1.1.3
Routing entry for 0.0.0.0/0
  Known via "kernel", distance 0, metric 0, best
  * 30.2.0.1, via eth1
```

```
=====agg.c1=====
=====success=====
```

```
agg.c1# show version
Quagga 0.99.23.1 ().
Copyright 1996-2005 Kunihiro Ishiguro, et al.
```

```
=====success=====
```

```
agg.c1# show ip route 11.1.1.3
Routing entry for 0.0.0.0/0
  Known via "kernel", distance 0, metric 0, best
  * 30.1.0.1, via eth1
```

Running Commands (Configure)

Configs/agg.c1:

```
conf t
hostname agg.c1
interface eth2
  ip add 20.1.0.3/24
  description links-to-core.1
  link-detect
  no shutdown
  exit
interface eth1
  ip add 30.1.0.2/24
  description links-to-agg.c1
  link-detect
  no shutdown
  exit

router ospf
  redistribute connected
  passive-interface eth0
  network 20.1.0.3/24 area 0
  network 30.1.0.2/24 area 0
  exit
end
write
```

Running Commands (Configure)

```
def get_config(devname):  
    with open("configs/{}.conf".format(devname)) as fh:  
        return fh.read()
```

```
async def demo():  
    devinfo = json.loads(open("devdb.json").read())  
    devices = [d['name'] for d in devinfo]
```

```
    dev_to_cmd = {}  
    for d in devices:  
        dev_to_cmd[dev_to_device(d)] = [get_config(d)]
```

```
    async with get_client() as client:  
        res = await client.bulk_run(dev_to_cmd)
```


Running Commands (Configure)

```
def get_config(devname):  
    with open("configs/{}.conf".format(devname)) as fh:  
        return fh.read()
```

```
async def demo():  
    devinfo = json.loads(open("devdb.json").read())  
    devices = [d['name'] for d in devinfo]
```

```
dev_to_cmd = {}  
for d in devices:  
    dev_to_cmd[dev_device(d)] = [get_config(d)]
```

```
async with get_client() as client:  
    res = await client.bulk_run(dev_to_cmd)
```

Running Commands (Configure)

```
def get_config(devname):  
    with open("configs/{}.conf".format(devname)) as fh:  
        return fh.read()
```

```
async def demo():  
    devinfo = json.loads(open("devdb.json").read())  
    devices = [d['name'] for d in devinfo]
```

```
    dev_to_cmd = {}  
    for d in devices:  
        dev_to_cmd[dev_to_device(d)] = [get_config(d)]
```

```
    async with get_client() as client:  
        res = await client.bulk_run(dev_to_cmd)
```

Running Commands (Configure)

```
=====agg.c1=====
=====success=====
agg.c1# show version
Quagga 0.99.23.1 ().
Copyright 1996-2005 Kunihiro Ishiguro, et al.
=====success=====
agg.c1# show ip route 11.1.1.3
Routing entry for 11.1.1.0/24
  Known via "ospf", distance 110, metric 20, best
  Last update 00:00:14 ago
  * 30.1.0.3, via eth1
=====agg.c2=====
=====success=====
agg.c2# show version
Quagga 0.99.23.1 ().
Copyright 1996-2005 Kunihiro Ishiguro, et al.
=====success=====
agg.c2# show ip route 11.1.1.3
Routing entry for 11.1.1.0/24
  Known via "ospf", distance 110, metric 30, best
  Last update 00:00:16 ago
  * 20.1.0.3, via eth2
```

Running Commands (Configure)

```
=====h1.r1.c1=====
=====success=====
netbot@h1:~$ traceroute -m 5 -w .5 11.2.1.3
traceroute to 11.2.1.3 (11.2.1.3), 5 hops max, 60 byte packets
 1 tor1.c1.eth1 (11.1.1.2) 0.060 ms 0.015 ms 0.009 ms
 2 agg.c1.eth1 (30.1.0.2) 0.031 ms 0.016 ms 0.026 ms
 3 agg.c2.eth2 (20.1.0.4) 0.044 ms 0.042 ms 0.034 ms
 4 tor1.c2.eth2 (30.2.0.3) 0.053 ms 0.041 ms 0.040 ms
 5 h1.r1.c2.eth1 (11.2.1.3) 0.059 ms 0.046 ms 0.048 ms
=====h1.r1.c2=====
=====success=====
netbot@h1:~$ traceroute -m 5 -w .5 11.1.1.3
traceroute to 11.1.1.3 (11.1.1.3), 5 hops max, 60 byte packets
 1 tor1.c2.eth1 (11.2.1.2) 0.044 ms 0.011 ms 0.008 ms
 2 agg.c2.eth1 (30.2.0.2) 0.029 ms 0.015 ms 0.012 ms
 3 agg.c1.eth2 (20.1.0.3) 0.035 ms 0.029 ms 0.017 ms
 4 tor1.c1.eth2 (30.1.0.3) 0.038 ms 0.022 ms 0.022 ms
 5 h1.r1.c1.eth1 (11.1.1.3) 0.042 ms 0.028 ms 0.029 ms
```

Setup FCR Service

- Easy to adapt
- Vendor information
- Device information

Setup FCR Service

Vendor Information

- For each vendor
 - Vendor name
 - Session type
 - Setup commands
 - Prompt_regex

Setup FCR Service

Vendor Information

- For each vendor
 - Vendor name: quagga
 - Session type: ssh
 - Setup commands: ["en", "term len 0"]
 - Prompt_regex: [\w.]+(\(config.*\))?\#\s*

Setup FCR Service

vendor_config.json

```
{
  "vendor_config": {
    "quagga": {
      "vendor_name": "quagga",
      "session_type": "ssh",
      "prompt_regex": ["[\\w.]+(\\(config.*\\))?#\\s*" ],
      "cli_setup": [ "en", "term len 0" ]
    },
    "debian": {
      "vendor_name": "debian",
      "session_type": "ssh",
      "prompt_regex": [ "\\w+@[\\w.-]+:[~^\\w-]+[#$]\\s*" ],
      "cli_setup": []
    }
  }
}
```


Setup FCR Service

Device Database: devdb.json

```
[  
  {"ip": "172.17.0.8", "name": "h1.r1.c2", "vendor": "debian"},  
  {"ip": "172.17.0.7", "name": "tor1.c2", "vendor": "quagga"},  
  {"ip": "172.17.0.6", "name": "agg.c2", "vendor": "quagga"},  
  {"ip": "172.17.0.5", "name": "h1.r1.c1", "vendor": "debian"},  
  {"ip": "172.17.0.4", "name": "tor1.c1", "vendor": "quagga"},  
  {"ip": "172.17.0.3", "name": "agg.c1", "vendor": "quagga"},  
  {"ip": "172.17.0.2", "name": "core.1", "vendor": "quagga"}  
]
```

Setup FCR Service

Device Database: Load Device information

```
class DeviceDB(BaseDeviceDB):

    def _create_device(self, name, addr, vendor, role):
        return DeviceInfo(service=self.service, hostname=name,
                           pref_ips=[DeviceIP('mgmt', addr, True)],
                           vendor_data=self.service.vendors.get(vendor))

    async def _fetch_device_data(self, name_filter=None):
        devices = []
        with open("devdb.json") as fh:
            for dev in json.loads(fh.read()):
                dev = self._create_device(dev["name"], dev["ip"], dev["vendor"])
                devices.append(dev)
        return devices
```

Setup FCR Service

FCR Service Class

```
class FCRService(FcrServiceBase):  
  
    def __init__(self):  
        super().__init__("FCR")  
  
        self.vendors = DeviceVendors(self)  
        self.device_db = DeviceDB(self)  
        self.service = CommandServer(self)  
  
def main():  
    service = FCRService()  
    service.start()
```

Setup FCR Service

Finally start the FCR service

```
(venv) devhost:~/nanog$ ./fcr_service.py --device_vendors device_vendors.json
```

```
INFO:fcr.DeviceVendors:loading local file
```

```
INFO:fcr.DeviceDB:Device data valid
```

```
INFO:FCR:Registering Counter manager
```

```
INFO:fcr.CommandServer:server started: 5000
```

Scale FCR service

- Add more instances as load increases.
 - Number of commands
 - How long are the commands active
- Load balance between these instances.

Summary

- FCR allows us to run command at Facebook scale
- Powerful APIs that are easy to use
- Service is easy to setup
- Easy to scale

facebook