

Run Control Plans

DEMO

Prerequisites

Your laptop, with

- Docker
- Docker-compose
- `git clone -b march-rc0 https://github.com/Juravenator/DUNE-RC-RC.git`

Docker used to simulate a distributed environment

You can also run on your (one) laptop on bare metal

Start docker images

```
# build local base images
make docker.images
# start a cluster of said images
make docker.start
```

```
→ DUNE-RC-RC git:(march-rc0) ✖ make docker.start
docker-compose -f docker/docker-compose.yml up
Creating docker_dune-rc-march-ru-2_1 ... done
Creating docker_dune-rc-march-gp-2_1 ... done
Creating docker_dune-rc-march-ru-3_1 ... done
Creating docker_dune-rc-march-gp-1_1 ... done
Creating docker_dune-rc-march-raft-2_1 ... done
Creating docker_dune-rc-march-raft-1_1 ... done
Creating docker_dune-rc-march-ru-1_1 ... done
Creating docker_dune-rc-march-gp-3_1 ... done
Creating docker_dune-rc-march-raft-3_1 ... done
Attaching to docker_dune-rc-march-ru-1_1, docker_dune-rc-mar
```

Set up your cluster

Ansible installs dependencies, and boots the control plane

```
→ DUNE-RC-RC git:(march-rc0) ✗ make docker.ansible
docker-compose -f docker/docker-compose.yml run --rm ansible -i /mnt/ansible/hosts.yaml /mnt/ansible/playbook.yaml
Creating docker_ansible_run ... done
```

```
PLAY [all] *****
```

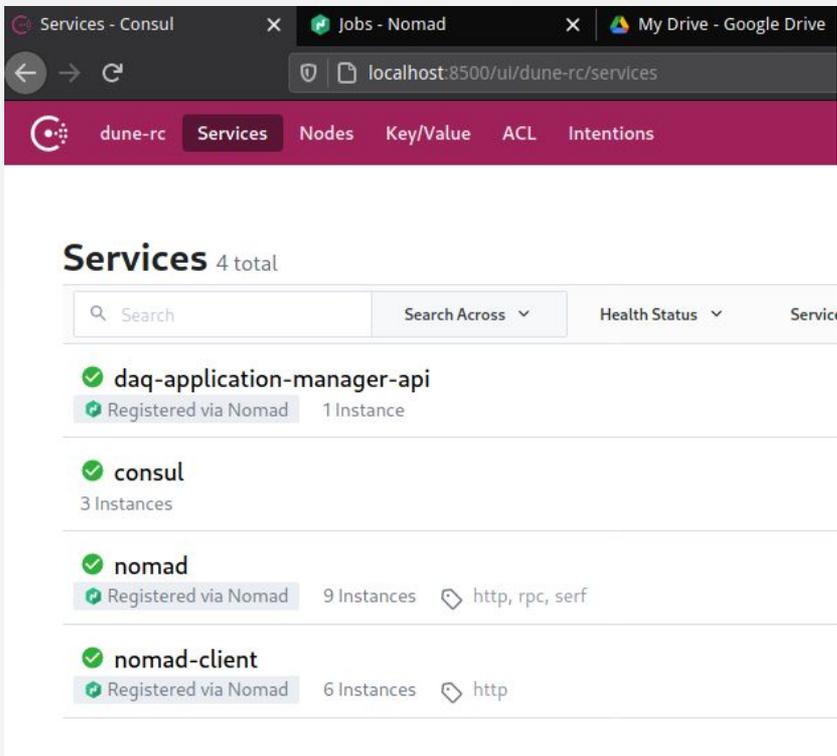
```
TASK [Gathering Facts] *****
```

```
ok: [dune-rc-march-gp-1]
ok: [dune-rc-march-ru-3]
ok: [dune-rc-march-gp-2]
ok: [dune-rc-march-ru-1]
ok: [dune-rc-march-ru-2]
ok: [dune-rc-march-gp-3]
ok: [dune-rc-march-raft-1]
ok: [dune-rc-march-raft-2]
ok: [dune-rc-march-raft-3]
```

```
TASK [yum] *****
```

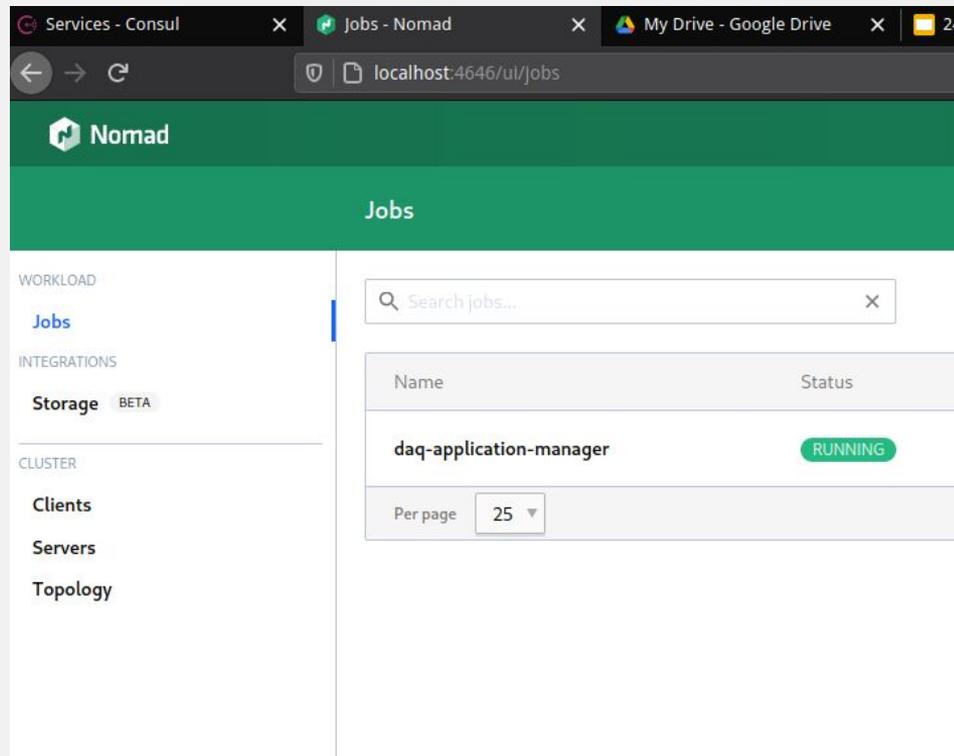
```
ok: [dune-rc-march-ru-3]
ok: [dune-rc-march-gp-2]
ok: [dune-rc-march-ru-2]
ok: [dune-rc-march-gp-1]
```

You now have a working RC cluster



The screenshot shows the Consul web interface at localhost:8500. The navigation bar includes 'dune-rc', 'Services', 'Nodes', 'Key/Value', 'ACL', and 'Intentions'. The main heading is 'Services 4 total'. Below this is a search bar and filters for 'Search Across' and 'Health Status'. A list of services is displayed, each with a green checkmark icon indicating it is healthy.

Service Name	Registered via Nomad	Instances	Ports
daq-application-manager-api	Registered via Nomad	1 Instance	
consul		3 Instances	
nomad	Registered via Nomad	9 Instances	http, rpc, serf
nomad-client	Registered via Nomad	6 Instances	http



The screenshot shows the Nomad web interface at localhost:4646. The navigation bar includes the Nomad logo and the heading 'Jobs'. The left sidebar contains navigation links for 'WORKLOAD', 'INTEGRATIONS', and 'CLUSTER'. The main content area shows a search bar for jobs and a table listing the current jobs.

Name	Status
daq-application-manager	RUNNING

Additional details visible in the interface include a 'Per page' dropdown set to 25 and a 'Storage BETA' integration link.

Run a DAQ application

march-rc0 ▾ [DUNE-RC-RC](#) / [configs](#) / [examples](#) / [working-daq-app](#) / [add.sh](#)

 **Glenn Dirx** cleaned up daq app json Late

 0 contributors

Executable File | 24 lines (19 sloc) | 960 Bytes

```
1  #!/usr/bin/env bash
2  set -o errexit -o nounset -o pipefail
3  IFS=$'\n\t\v'
4  cd `dirname "${BASH_SOURCE[0]:-$0}"`
5
6  # there is no constraint that these resources are added in any particular order
7  # but the order presented here makes the most sense
8
9  # to demonstrate how services work, the example config includes a reference to
10 # a service called 'datastorage'
11 # register this service (stolen from another example) to see it resolve
12 ../external_service/add.sh
13
14 # schedule a daq application with nomad (process manager & scheduler)
15 curl -X POST --fail --data @ru-job.json http://localhost:4646/v1/jobs
16 echo ""
17
18 # put the desired daq config template in the raft key-value store
19 curl -X PUT --fail --data @my-first-config.json 'http://localhost:8500/v1/kv/daq-applications/configs/my-first-config'
20 echo ""
21
22 # register a config with daq-app-manager
23 curl -X PUT --fail --data @my-first-daq-app.json 'http://localhost:8500/v1/kv/daq-applications/my-first-daq-app'
24 echo ""
```

Glenn Dirx examples

La

0 contributors

43 lines (37 sloc) 810 Bytes

```

1 job "daq-ru-pu-52-51" {
2   datacenters = ["dune-rc"]
3
4   type = "service"
5
6   update {
7     max_parallel = 1
8   }
9
10  group "daq-ru" {
11
12    network {
13      port "api" {}
14    }
15
16    task "daq-application" {
17      driver = "raw_exec"
18      config {
19        command = "bash"
20        args = ["/dune-rc/hacks/daq-app-starter.sh", "--commandFacility", "rest://localhost:${NOMAD_PORT_api}"]
21      }
22      constraint {
23        attribute = "${meta.module-a-pu-apas}"
24        set_contains = "42-51"
25      }
26      constraint {
27        attribute = "${meta.cvmfs}"
28        value = "/cvmfs"
29      }
30    }
31
32    service {
33      name = "daq-ru-pu-52-51-api"
34      port = "api"
35      // check {
36      //   type = "http"
37      //   path = "/health"
38      //   interval = "10s"
39      //   timeout = "2s"
40      // }
41    }
42  }
43 }

```

Glenn Dirx cleaned up daq app json

0 contributors

38 lines (38 sloc) 1.02 KB

```

1 {
2   "Job": {
3     "Datacenters": ["dune-rc"],
4     "ID": "daq-ru-pu-52-51",
5     "Name": "daq-ru-pu-52-51",
6     "TaskGroups": [
7       {
8         "Migrate": {"MaxParallel": 1},
9         "Name": "daq-ru",
10        "Networks": [{"DynamicPorts": [{"Label": "api"}]},
11        "Services": [{"Name": "daq-ru-pu-52-51-api", "PortLabel": "api"}],
12        "Tasks": [
13          {
14            "Config": {
15              "command": "bash",
16              "args": [
17                "/dune-rc/hacks/daq-app-starter.sh",
18                "--commandFacility",
19                "rest://localhost:${NOMAD_PORT_api}"
20              ]
21            },
22            "Constraints": [
23              {
24                "LTarget": "${meta.module-a-pu-apas}",
25                "Operand": "set_contains",
26                "RTarget": "42-51"
27              },
28              {"LTarget": "${meta.cvmfs}", "Operand": "=", "RTarget": "/cvmfs"}
29            ],
30            "Driver": "raw_exec",
31            "Name": "daq-application"
32          }
33        ]
34      }
35    ],
36    "Type": "service"
37  }
38 }

```

 march-rc0 ▾[DUNE-RC-RC](#) / [configs](#) / [examples](#) / [working-daq-app](#) / **my-first-daq-app.json****Glenn Dirkx** examples 0 contributors

14 lines (14 sloc) | 296 Bytes

```
1  {
2    "meta": {
3      "kind": "daq-application",
4      "name": "my-first-daq-app",
5      "owner": ""
6    },
7    "spec": {
8      "daq-service": "daq-ru-pu-52-51-api",
9      "run-number": "123",
10     "configkey": "/daq-applications/configs/my-first-config",
11     "enabled": true,
12     "desired-state": "running"
13   }
14 }
```

```
→ DUNE-RC-RC git:(march-rc0) X configs/examples/working-daq-app/add.sh
registering:
true
getting:
[{"ID":"","Node":"fake","Address":"","Datacenter":"dune-rc","TaggedAddresses":null,"M
e":{},"ServiceConnect":{},"CreateIndex":169,"ModifyIndex":169}]
{"EvalID":"890ec894-63cb-9918-99eb-19e2a69fdffa","EvalCreateIndex":37,"JobModifyIndex
true
true
```

Services - Consul x Jobs - Nomad x My Drive - Google 10

localhost:8500/ui/dune-rc/services

dune-rc Services Nodes Key/Value ACL Intentions

Services 6 total

Search Search Across Health Status

- datastorage 1 Instance
- ✓ daq-application-manager-api Registered via Nomad 1 Instance
- ✓ daq-ru-pu-52-51-api Registered via Nomad 1 Instance

Services - Consul x Jobs - Nomad x My Drive - Google Drive x 24-feb-2021 R

localhost:4646/ui/jobs

Nomad

Jobs

WORKLOAD

Jobs

INTEGRATIONS

Storage BETA

CLUSTER

Clients

Servers

Topology

Search jobs... X

Name	Status	Type	Priority	Groups	Summary
daq-ru-pu-52-51	RUNNING	service	50	1	<div style="width: 100%; height: 10px; background-color: green;"></div>
daq-application-manager	RUNNING	service	50	1	<div style="width: 100%; height: 10px; background-color: green;"></div>

Per page 25 1-2 of 2

march-rc0 ▾ DUNE-RC-RC / configs / examples / working-daq-app / status.sh

Glenn Dirx cleaned up daq app json

0 contributors

Executable File | 6 lines (5 sloc) | 200 Bytes

```

1 #!/usr/bin/env bash
2 set -o errexit -o nounset -o pipefail
3 IFS=$'\n\t\v'
4 cd `dirname "${BASH_SOURCE[0]:-$0}"`
5
6 curl 'http://localhost:8500/v1/kv/daq-applications/my-first-daq-app?raw=' 2>/dev/null | jq

```

```

→ DUNE-RC-RC git:(march-rc0) ✖ configs/examples/working-daq-app/status.sh
{
  "meta": {
    "kind": "daq-application",
    "name": "my-first-daq-app",
    "owner": ""
  },
  "spec": {
    "configkey": "/daq-applications/configs/my-first-config",
    "daq-service": "daq-ru-pu-52-51-api",
    "desired-state": "running",
    "enabled": true,
    "run-number": "123"
  },
  "status": {
    "commandpostfailed": false,
    "configkeyexists": true,
    "configrendered": true,
    "daqserviceexists": true
  }
}

```

→ ~ de docker_dune-rc-march-ru-1_1

[root@3f80208ad3ca /]# ps -aux

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	112952	7448	?	Ss	13:32	0:00	/usr/sbin/sshd -D
root	8344	0.3	0.2	785952	66512	?	Sl	13:37	0:03	/usr/bin/consul agent -config-dir=/etc/consul.d/
root	10406	0.3	0.2	1353068	70860	?	Sl	13:38	0:02	/usr/bin/nomad agent -config=/etc/nomad.d/config.hcl
root	12038	0.1	0.0	1133720	23488	?	Sl	13:46	0:00	/usr/bin/nomad logmon
root	12049	0.1	0.1	1431720	37092	?	Ssl	13:46	0:00	/usr/bin/nomad executor {"LogFile":"/opt/nomad/data/alloc
root	12060	0.1	0.0	235176	22688	?	Sl	13:46	0:00	daq_application --commandFacility rest://localhost:25241
root	14868	0.0	0.0	11844	2984	pts/0	Ss	13:52	0:00	bash
root	14885	0.0	0.0	51748	3424	pts/0	R+	13:53	0:00	ps -aux

[root@3f80208ad3ca /]#

Next Steps

- Implement the HTTP API return path logic in daq-app-manager
- Make config generation generate configs like shown today
- Implement controller for higher-level config (partition level)
- CLI (now we interact with bare REST APIs)