O'REILLY

OSCON

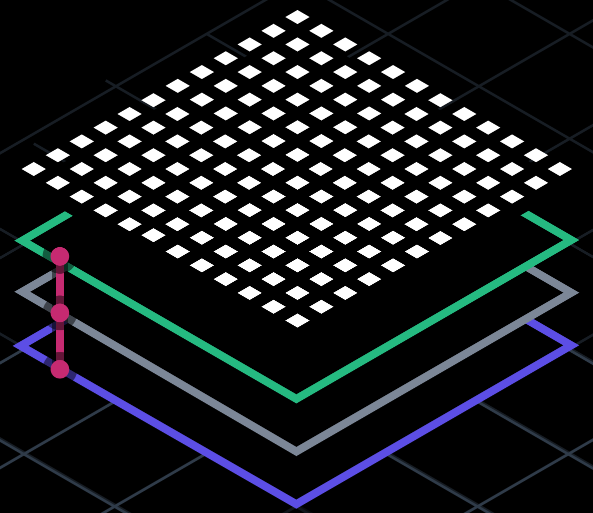
HashiCorp

DEVOPS DELIVERED

HashiCorp

Nomad

WELCOME TO NOMAD HANDS-ON TRAINING!



oscon.com

#oscon

Anubhav Mishra

Developer Advocate, HashiCorp
@anubhavm

HashiCorp

Nomad

Agenda

1. Architecture Overview
2. Installing and Configuring Nomad
3. Creating and Running Jobs
4. Service Discovery with Consul
5. Operating Nomad
6. Interacting with the HTTP API
7. Advanced Job Strategies

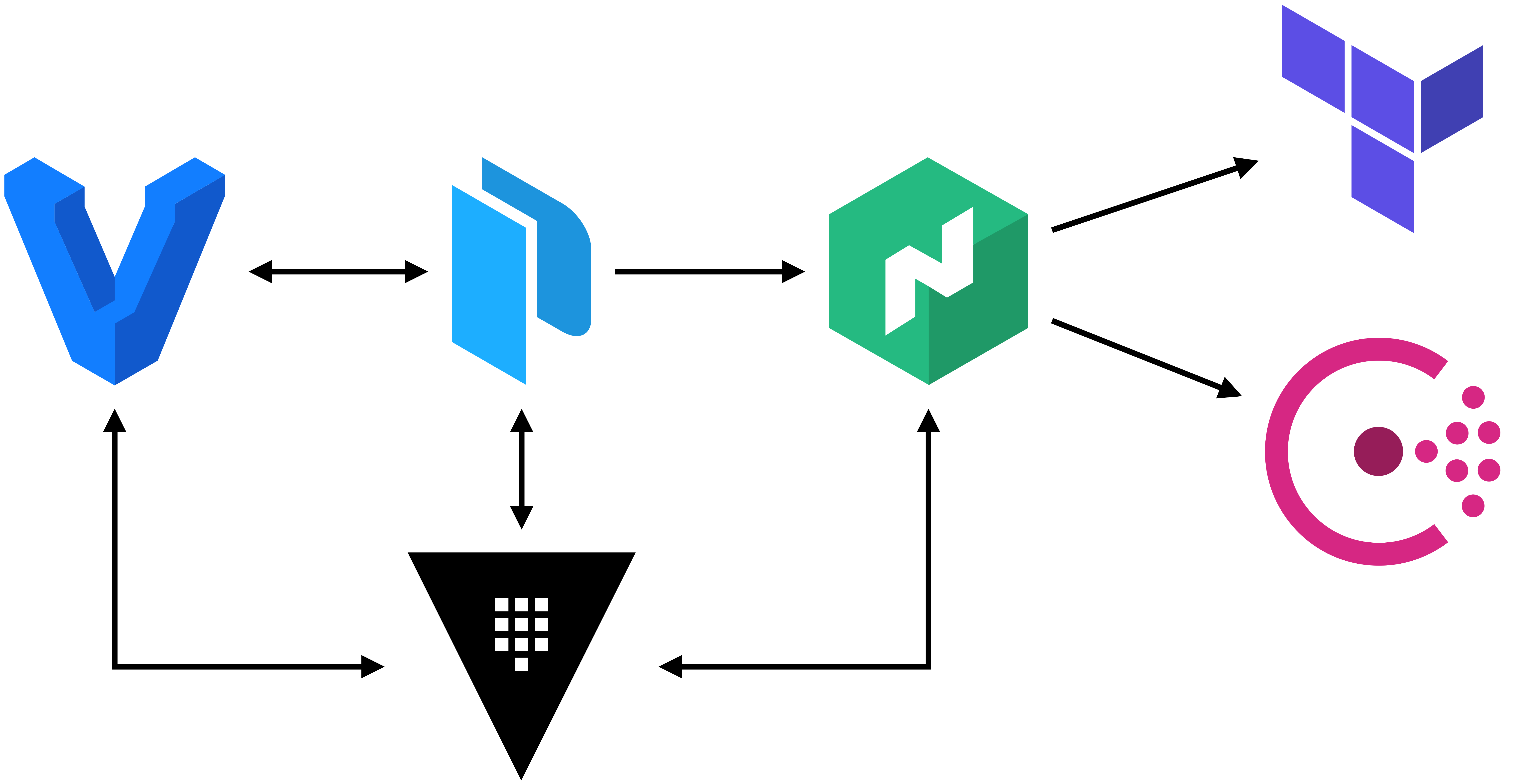
Workstations

Claim your workstation!

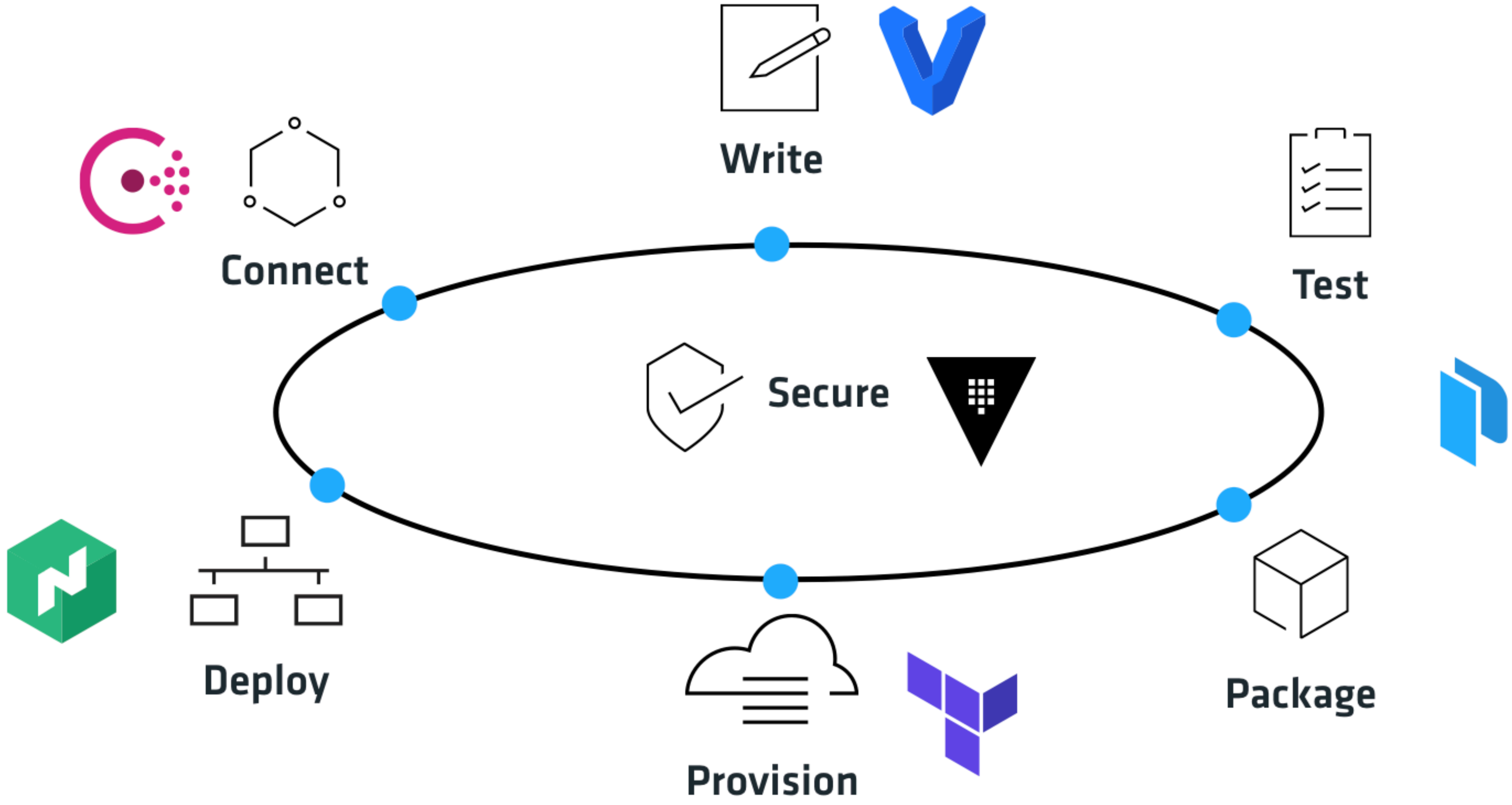
https://hashi.co/nomad-hands-on-oscon

Introduction

Nomad Powers Application Deployment



Application Delivery Lifecycle



Nomad's Goals

Treat entire data center as a collection of resources

Support non-containerized workloads

Achieve massive scale and speed

Cross-platform portability

Support heterogeneous workflows

Nomad

Million Container Challenge

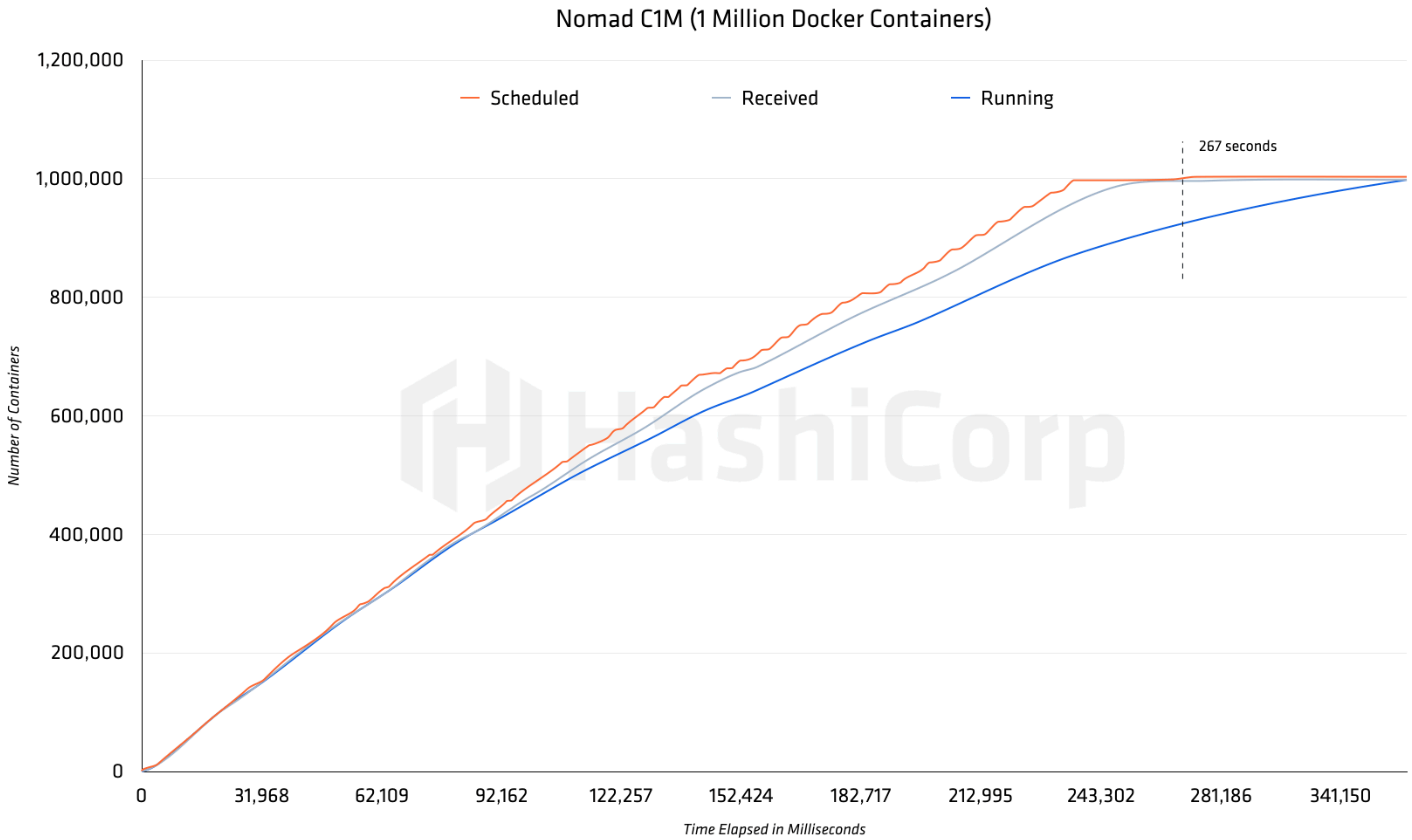
1,000 Jobs

1,000 Tasks per Job

5,000 Hosts on GCE

1,000,000 Containers

```
==> Nomad agent started! Log data will stream in below:

    2016/03/14 17:34:26 [DEBUG] client: avail  ble drivers [raw_exec java qemu docker exec]
    2016/03/14 17:34:26 [INFO] client: s          er address list: [127.0.0.1:4647]
    2016/03/14 17:34:26 [DEBUG] client:            checking for node changes at duration 5s
    2016/03/14 17:34:27 [DEBUG]                  mplete
    2016/03/14 17:34:28 [DEBU
    2016/03/14 17:35:05                             8 (pulled 1) (filtered 0)
    2016/03/14 17:3                                dated 0) (ignore 0)
    2016/03/14 17:                                76c 4259-a20f-1581-877f-1a
    2016/0                                         '76cb4259-a20f-158
    2016                                            (filtered 0)
                                                        0)
                                                       99-94
                                                       8-975
```



Nomad C1M (1 Million Docker Containers)
Scheduled
Received
Running
267 seconds
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Number of Containers
0
31,968
62,109
92,162
122,257
152,424
182,717
212,995
243,302
281,186
341,150
Time Elapsed in Milliseconds

“640 KB ought to be enough for anybody.”

- Bill Gates

2nd Largest Hedge Fund

18K Cores

5 Hours

2,200 Containers/second

CITADEL

Nomad vs. Other Tools

Schedule multiple workload types: VMs, containers, binaries, etc

Optimistically concurrent to schedule faster and at greater scale

Globally-aware with multi-region and multi-datacenter support

Operationally simple

Integration into the HashiCorp ecosystem

Glossary

Glossary

Node

Physical or virtual machine in the cluster. In the context of Nomad, a node is a machine running the Nomad agent.

Glossary

Agent

Long-running daemon on every member of the Nomad cluster. The agent is able to run in either client or server mode.

Glossary

Agent (Client)

Agent that fingerprints the host to determine capabilities, resources, and available drivers.

Glossary

Agent (Server)

Agent that holds the global state of the cluster and participates in scheduling decisions.

Glossary

Job

Definition of how a workload should be scheduled. The job specification is composed of one or more task groups, and each task defines a series of resource constraints and configuration.

Glossary

Job File

HCL or JSON configuration file on disk which describes how a workload should be scheduled.

Glossary

Driver

Pluggable components that execute a task and provide resource isolation. Example drivers include *docker*, *java*, and *raw-exec*.

Glossary

Task

A command, service, application or "set of work" to execute. Tasks are executed by their driver.

Glossary

Task Group

A collection of individual tasks that should be co-located on the same node. This is especially useful for applications that require low latency or have high throughput to another application.

Glossary

Evaluation

A calculation performed by the Nomad servers to determine what action(s) need to take place to execute a job.

Glossary

Allocation

An instance of a task group placed on a node. Allocations can fail (not enough memory, node is down, etc).

Glossary

Datacenter

Networking environment that is private, low latency, and high bandwidth. Example datacenters include the `us-east-1` or `us-west-1`.

Glossary

Region

Collection of multiple datacenters, typically grouped geographically. For example, the `north-america` region might include the `us-east-1` and `us-west-1` datacenters.

Glossary

Consensus

Agreement upon the elected leader.

Glossary

Gossip

Random node-to-node communication primarily over UDP that provides membership, failure detection, and event broadcast information to the cluster. Built on Serf.

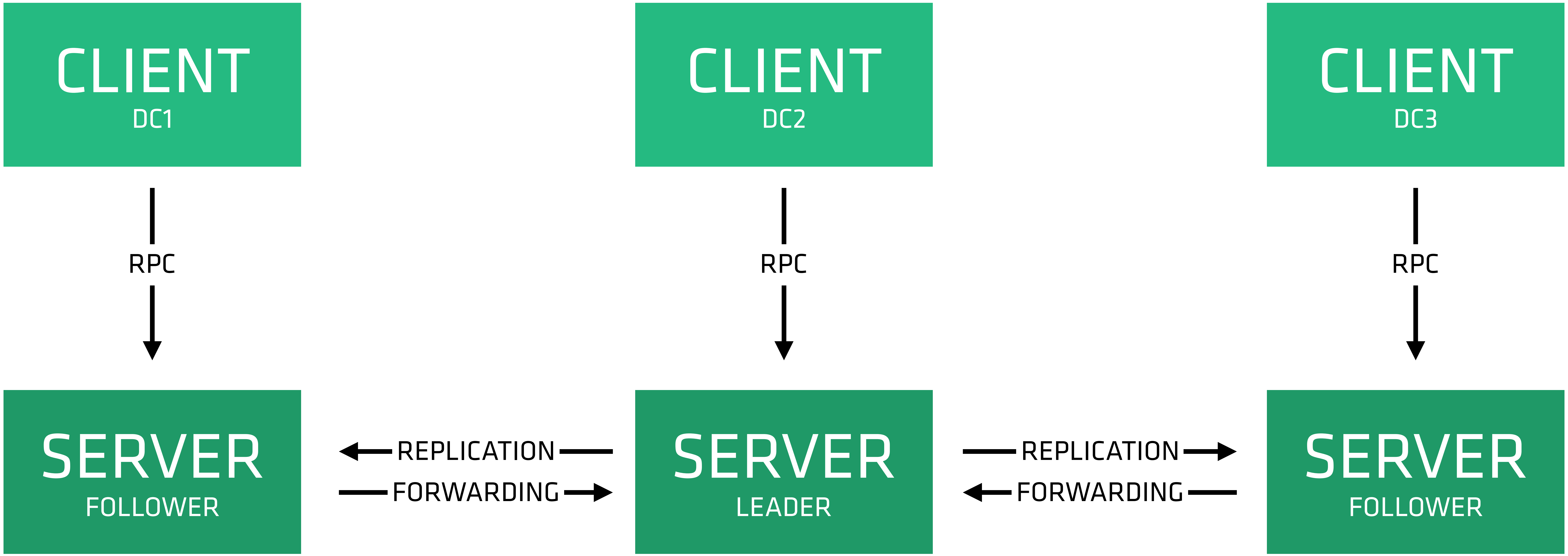
Glossary

Bin-Packing

A algorithm which optimizes the resource utilization and density of applications, but is also augmented by affinity and anti-affinity rules.

Architecture

Single Region Architecture



HashiCorp

Multi-Region Architecture

REGION A
SERVER
FOLLOWER
REPLICATION
SERVER
LEADER
REPLICATION
FORWARDING
SERVER
FOLLOWER
GOSSIP
REGION FORWARDING
REGION B
SERVER
FOLLOWER
REPLICATION
SERVER
LEADER
REPLICATION
FORWARDING
SERVER
FOLLOWER

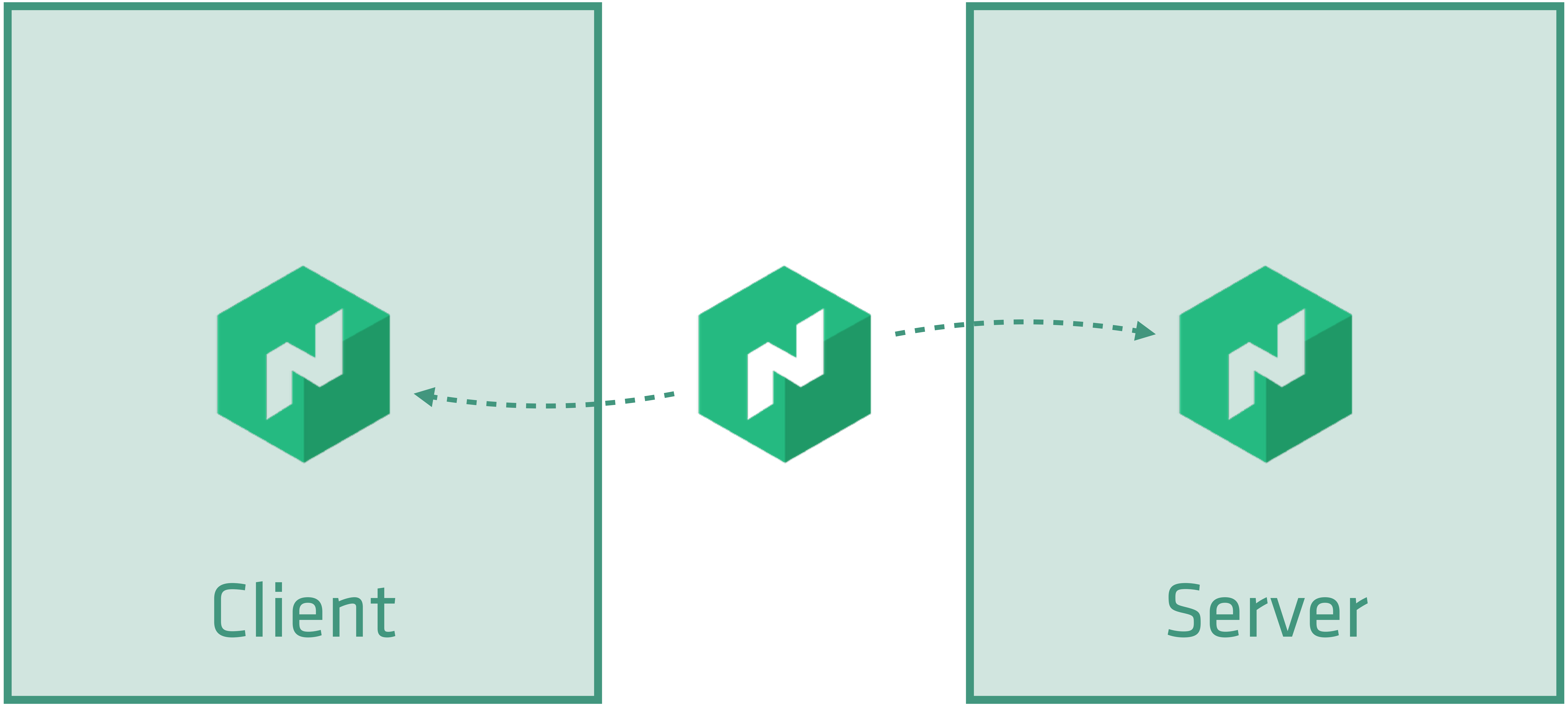
100’s of Regions

10,000’s of Clients per Region

1000’s of Jobs per Region

Installing & Configuring

Single Binary



Agent Functionality (Client)

Fingerprint to determine resources and capabilities of each host

Send node information to the central server cluster

Heartbeat to provide liveness

Run any tasks assigned by the central server cluster

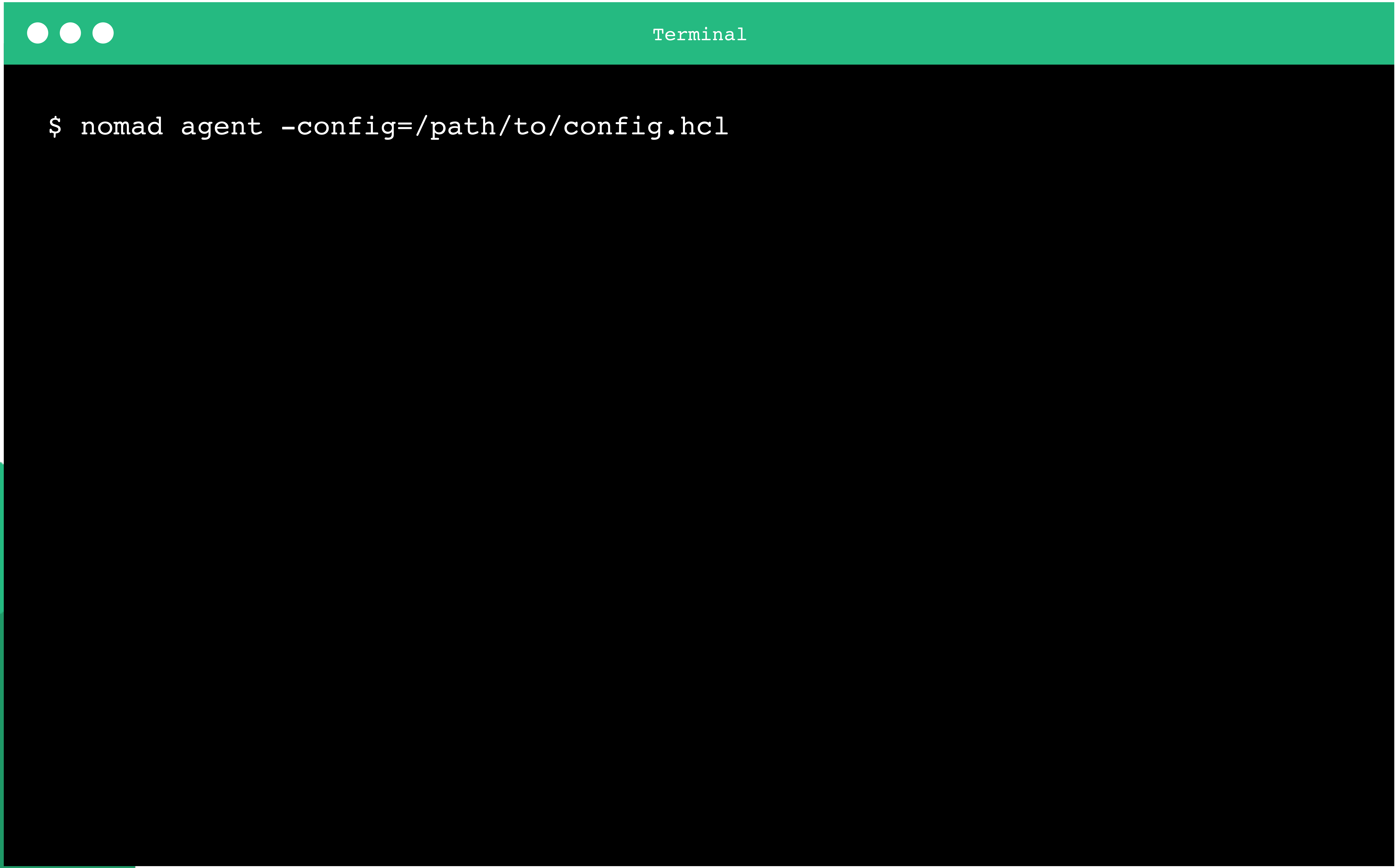
Agent Functionality (Server)

Store resource, capability, and availability of each host

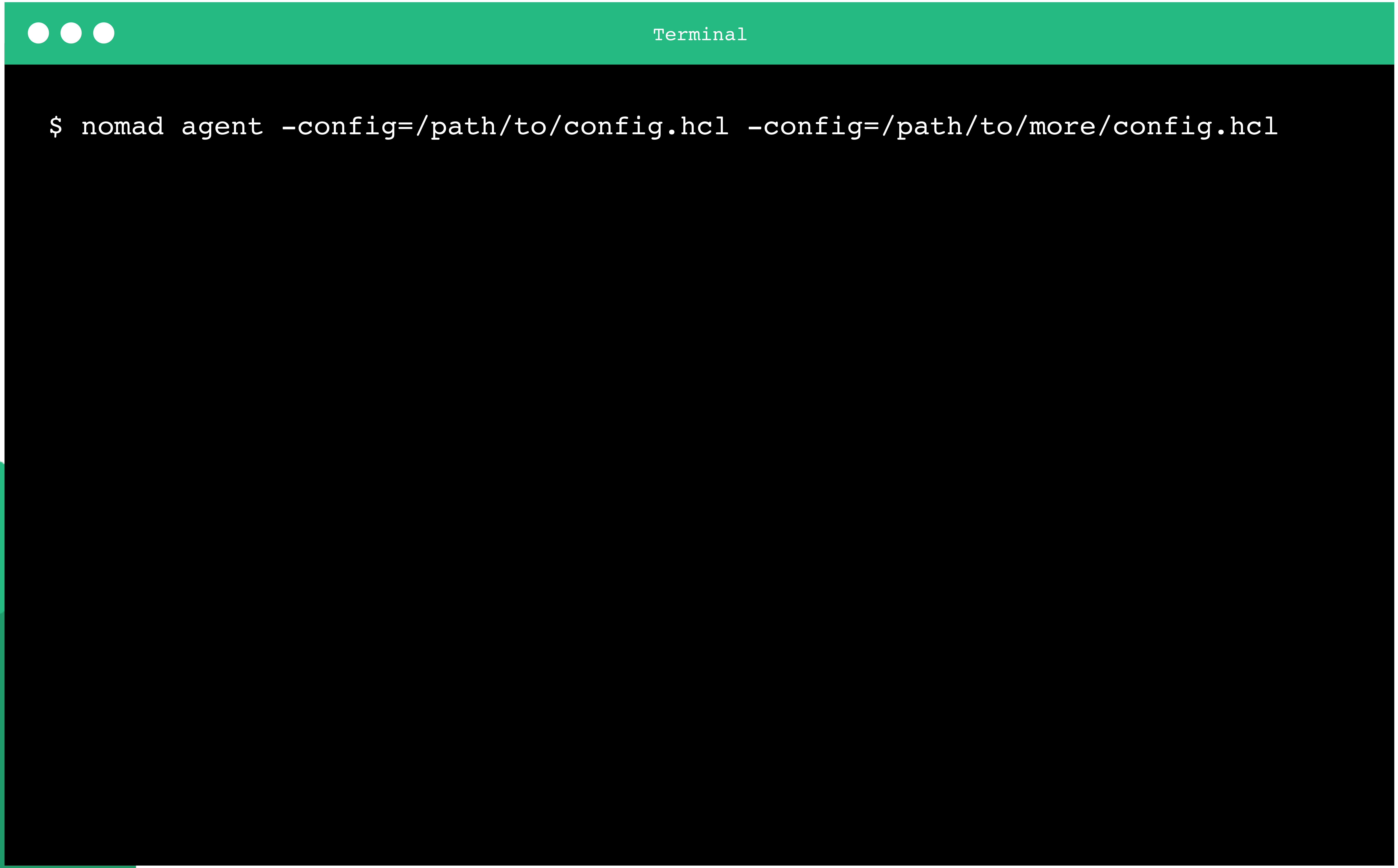
Schedule workloads

Participate in leader election and state replication (consensus)

Discover other Nomad servers (gossip)



```
$ nomad agent -config=/path/to/config.hcl
```



```
$ nomad agent -config=/path/to/config.hcl -config=/path/to/more/config.hcl
```

Bootstrapping Nomad

Bootstrapping Nomad

There are two common strategies for bootstrapping a Nomad cluster: *automatic* and *manual*.

We will walk through the manual steps first, then automatic.

Bootstrapping Nomad: Manual

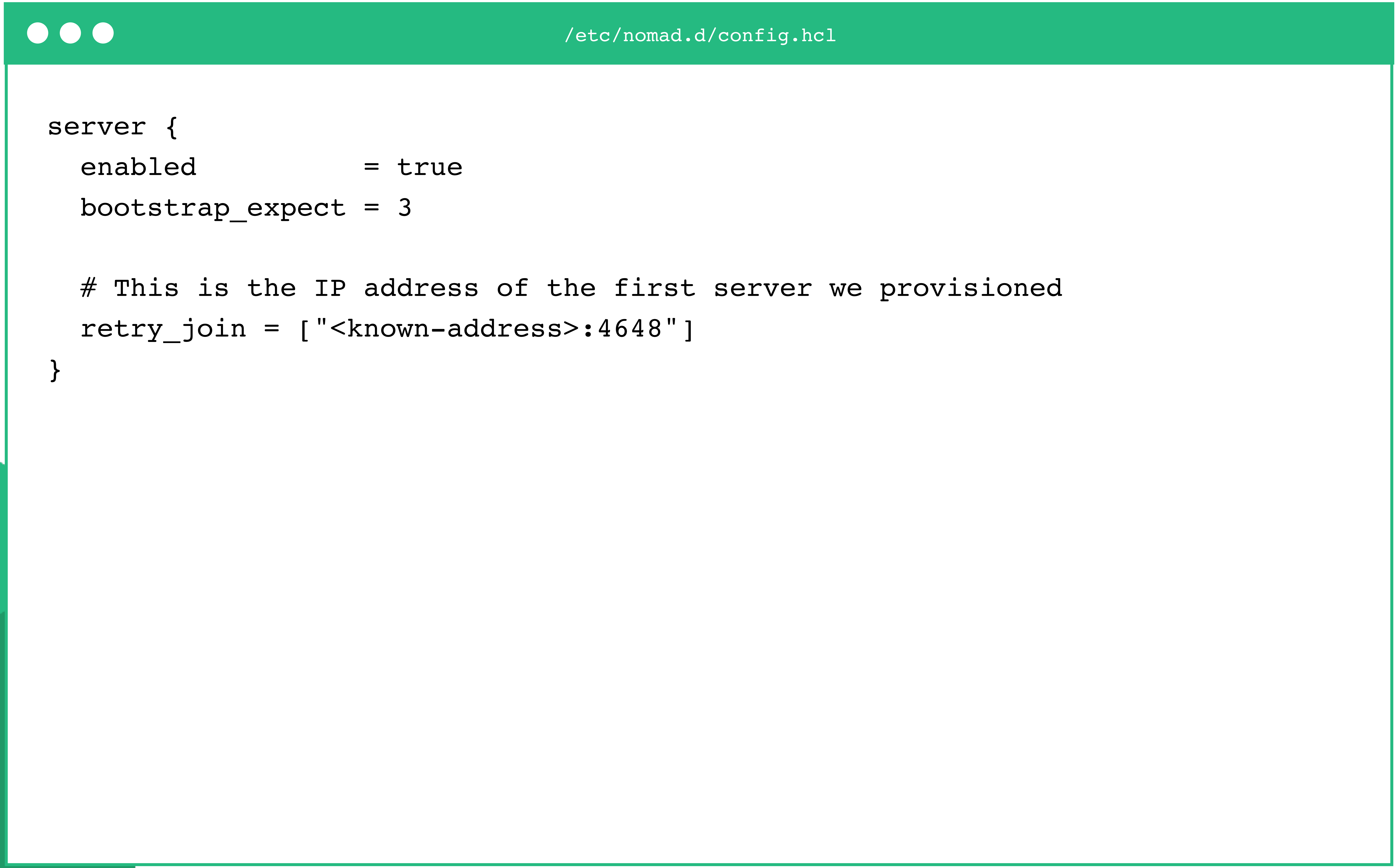
Chicken-and-egg problem: requires one server IP

The `bootstrap_expect` field tells how many servers to reach quorum

This server IP is specified in a configuration file or used with the `nomad server join` command

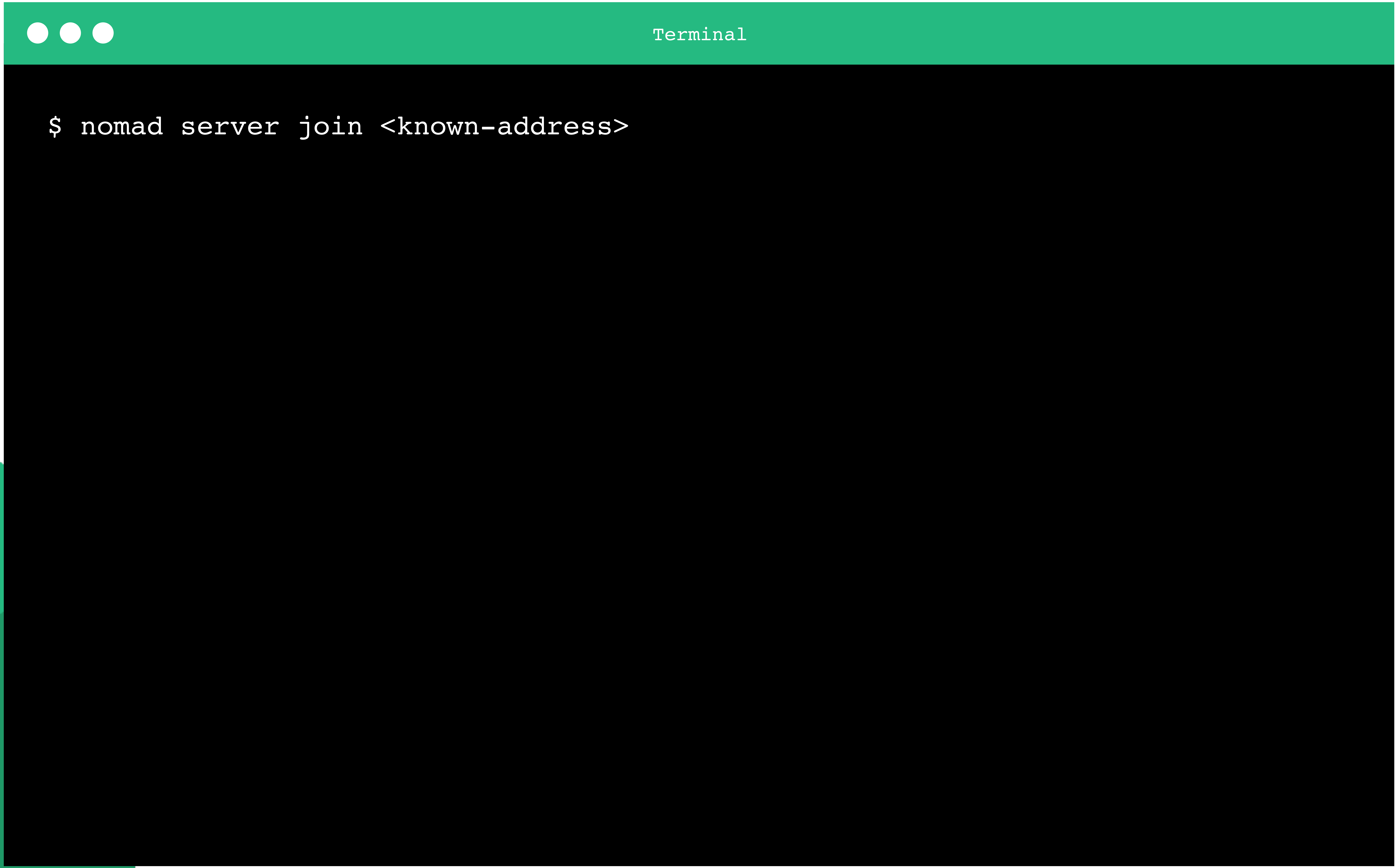
Clients specify server IPs via configuration file

Requires human involvement

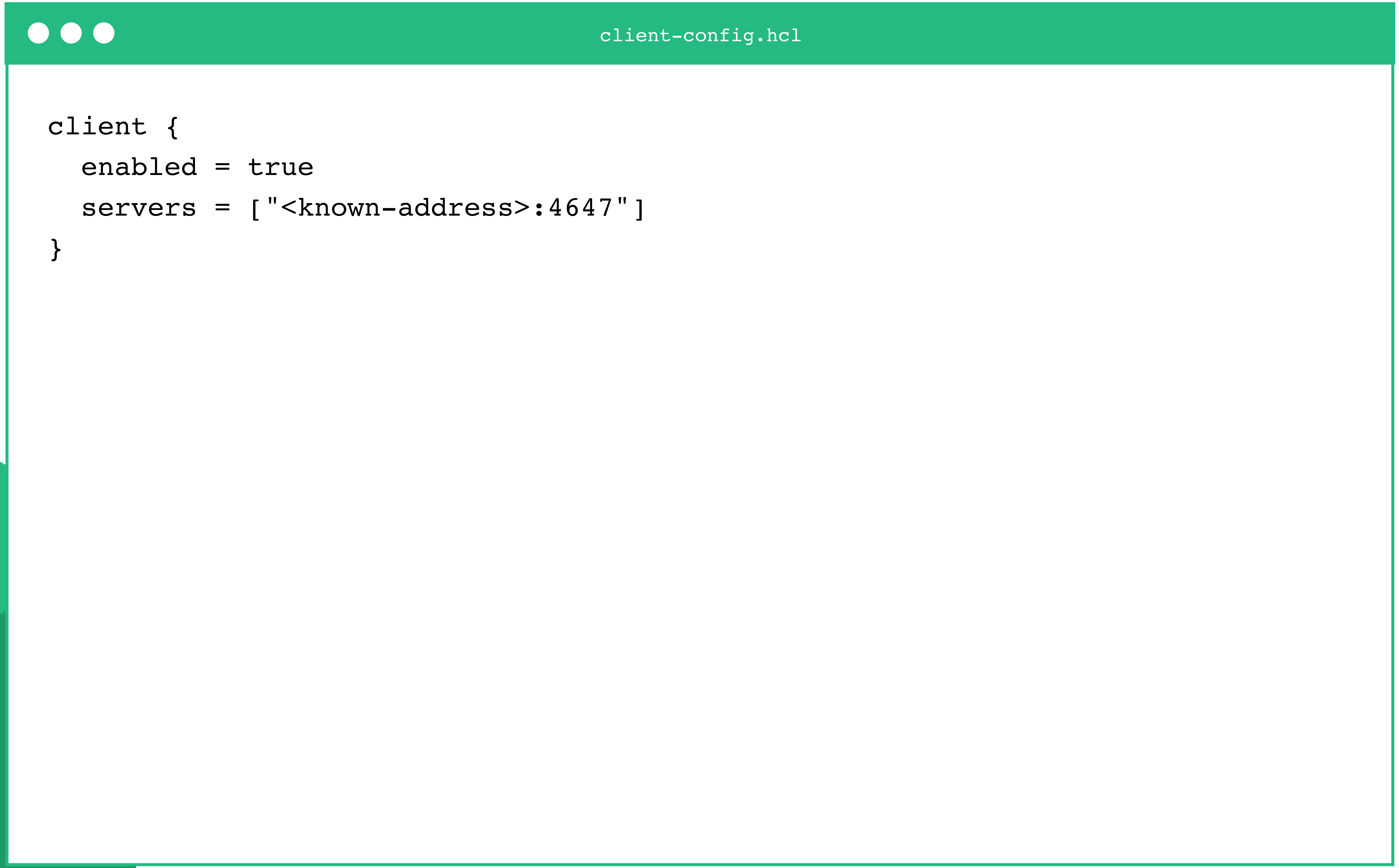


/etc/nomad.d/config.hcl

```
server {
  enabled          = true
  bootstrap_expect = 3

  # This is the IP address of the first server we provisioned
  retry_join = ["<known-address>:4648"]
}
```



```
$ nomad server join <known-address>
```



```
client {
  enabled = true
  servers = ["<known-address>:4647"]
}
```

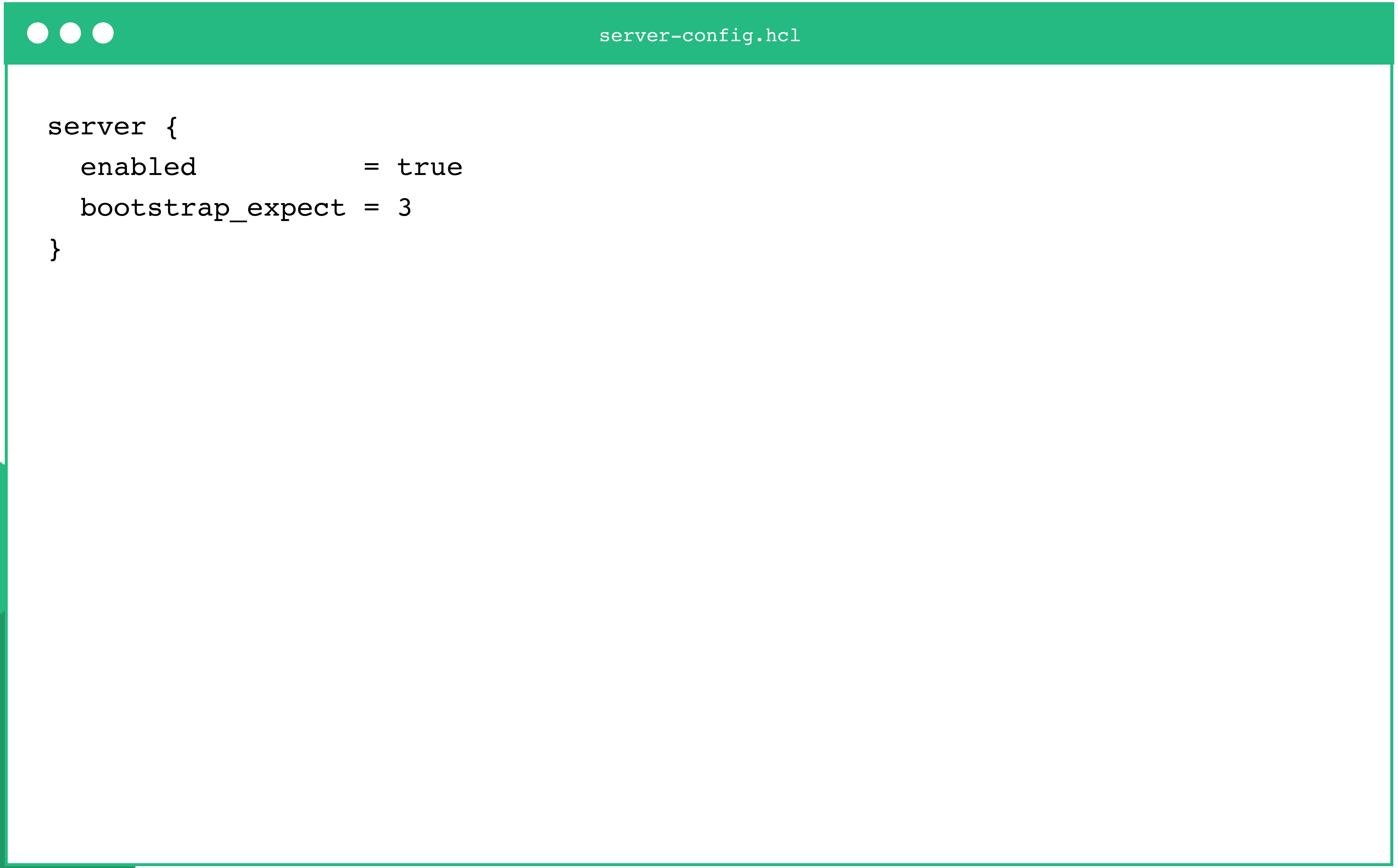
Bootstrapping Nomad: Automatic

Leverages another open source tool - Consul

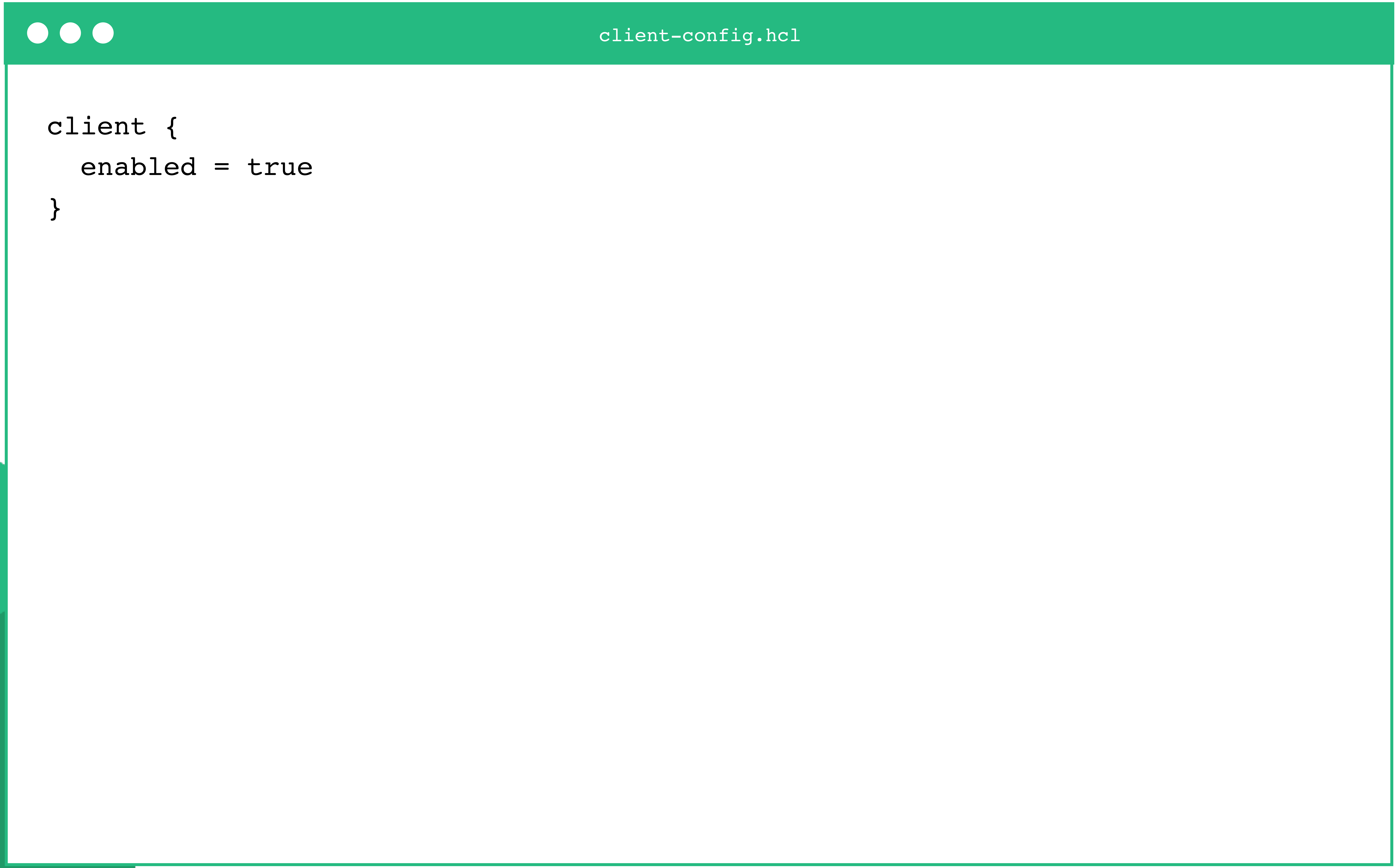
The `bootstrap_expect` field tells how many servers to reach quorum

Nomad assumes the Consul agent is accessible on the local IP and port, but this is configurable

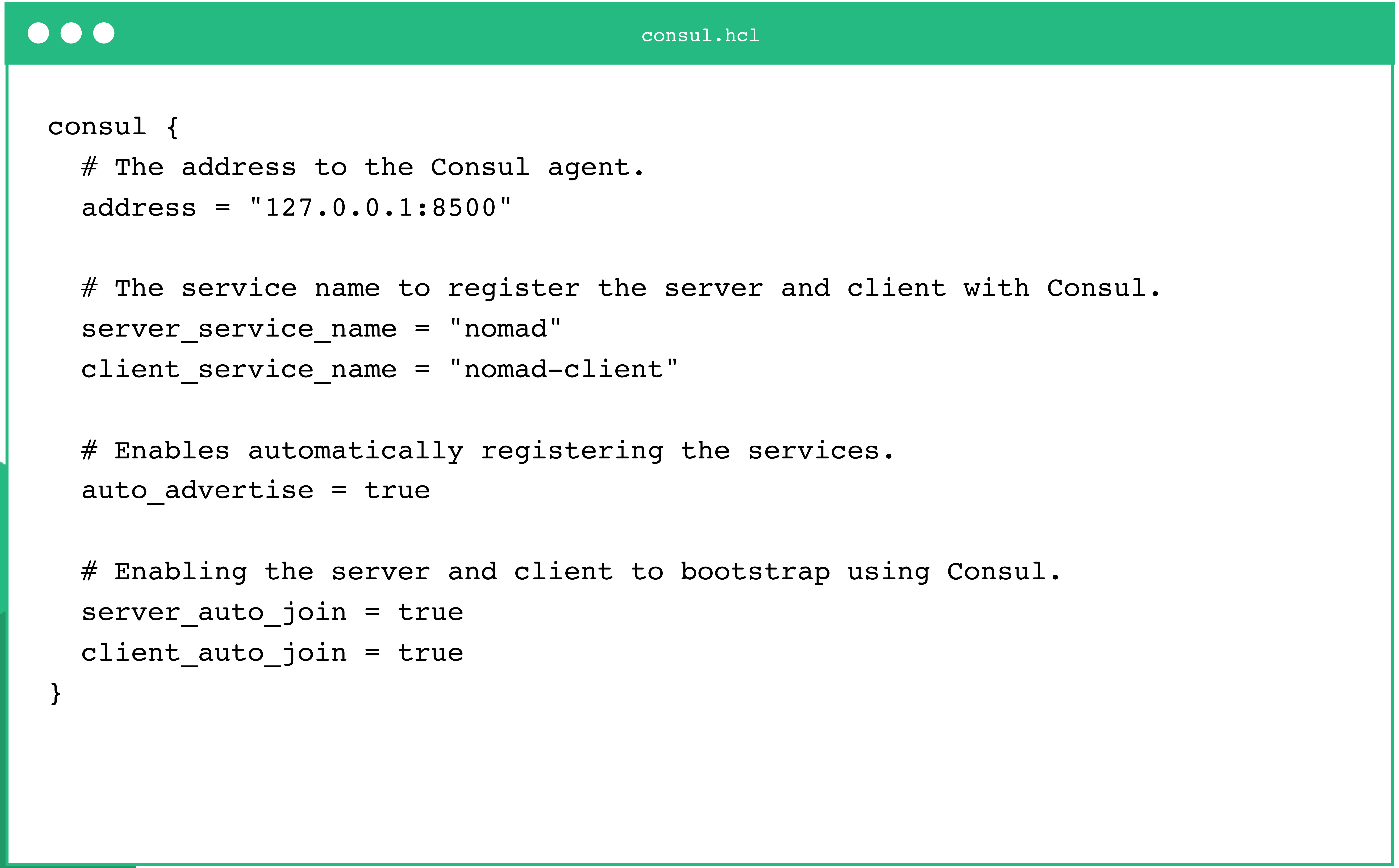
Fully automated and scalable



```
server {
  enabled          = true
  bootstrap_expect = 3
}
```



```
client {
  enabled = true
}
```



```
consul {
  # The address to the Consul agent.
  address = "127.0.0.1:8500"

  # The service name to register the server and client with Consul.
  server_service_name = "nomad"
  client_service_name = "nomad-client"

  # Enables automatically registering the services.
  auto_advertise = true

  # Enabling the server and client to bootstrap using Consul.
  server_auto_join = true
  client_auto_join = true
}
```

Exercise: Connect to Workstation

Go to: https://hashi.co/nomad-hands-on-oscon and claim a workstation IP.

SSH into your workstation using the provided credentials.

```
ssh nomaduser@<your.ip.address>
password: oscon2018
```

Change directory into `/workstation/nomad`.

Creating & Running Jobs

Nomad Jobs

Jobs specifications are defined in HCL or JSON, but we will use HCL for this training.

An example can be generated by running `nomad init`.

Nomad Job Types

Nomad has three scheduler types that can be used when creating your job: *service*, *batch*, and *system*.

Nomad Job Types: Service

Service Scheduler Job Type

The service scheduler is designed for scheduling long-lived services that should never go down. The service scheduler ranks a large portion of the nodes that meet the jobs constraints and selects the optimal node to place a task group on.

Examples: *webapp*, *redis*

Nomad Job Types: Batch

Batch Scheduler Job Type

Batch jobs are less sensitive to short-term performance fluctuations and are short lived, finishing after some period.

Examples: *billing*, *data replication*

Nomad Job Types: System

System Scheduler Job Type

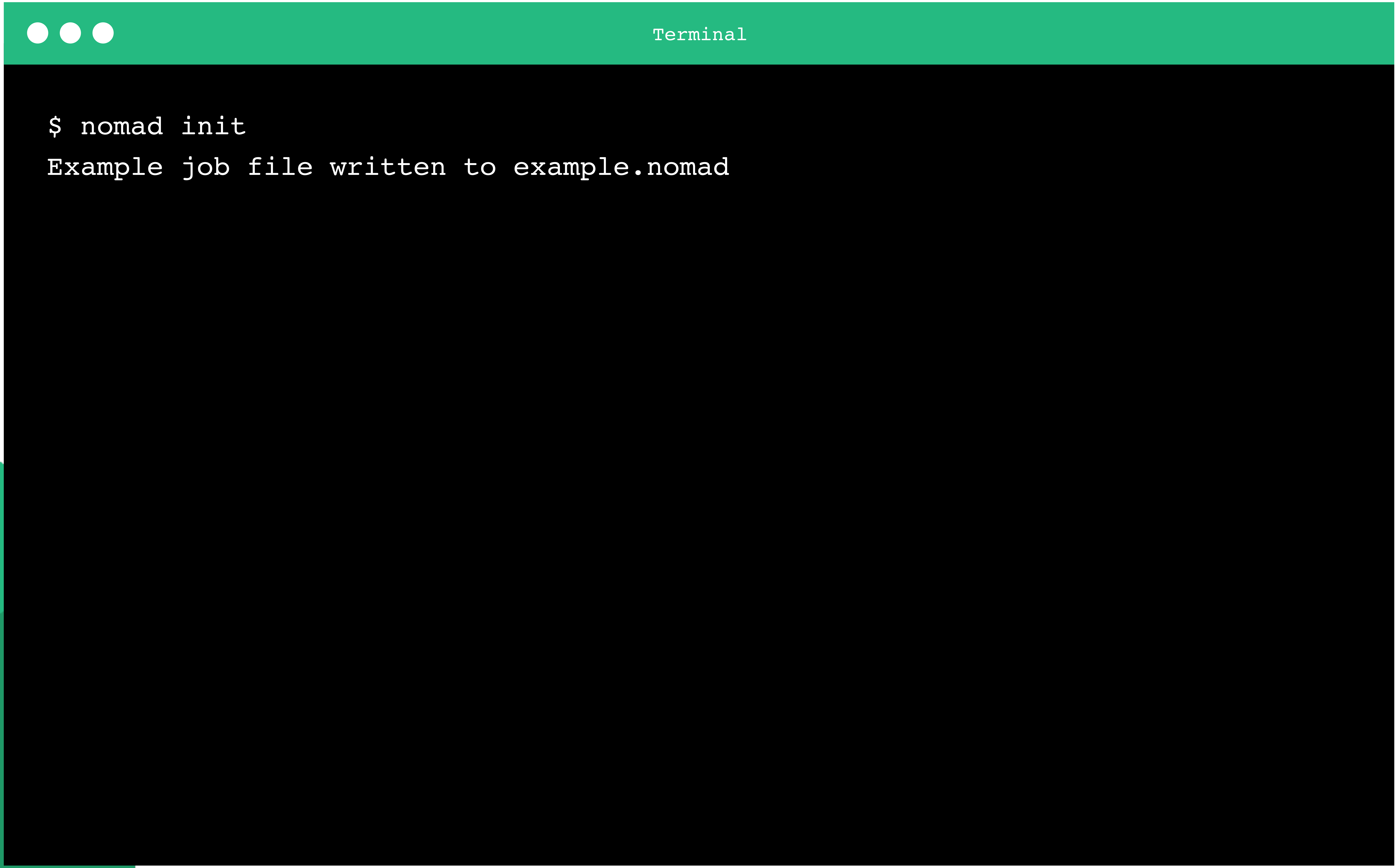
The system scheduler is used to register jobs that should be run on all clients that meet the job's constraints. The system scheduler is also invoked when clients join the cluster or transition into the ready state.

Examples: *logging agent*, *security auditing tool*

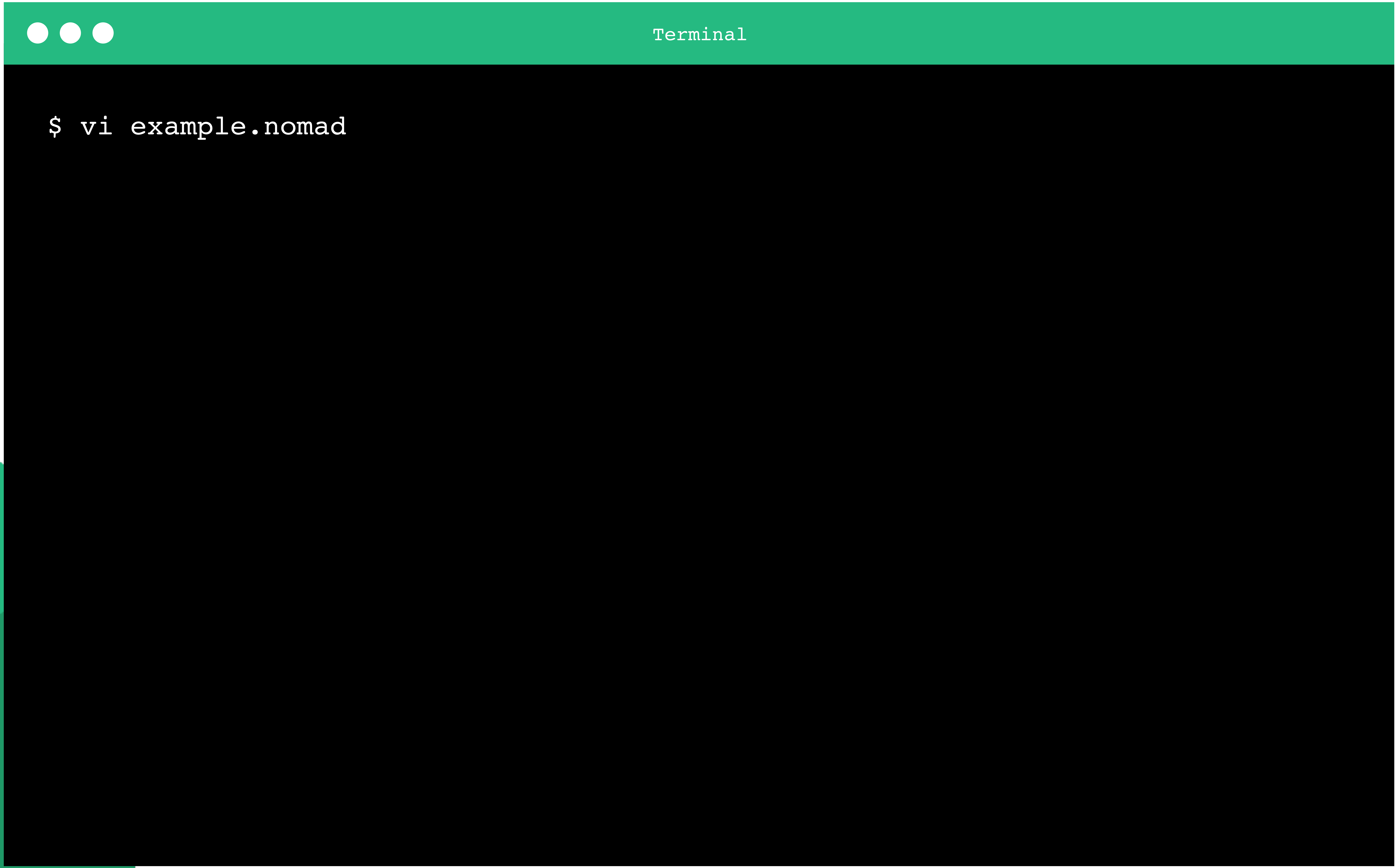
Exercise: Run `nomad init`

Inside the `/workstation/nomad` folder, run the `nomad init` command to generate a sample job file.

Open the resulting file on the workstation in an editor.



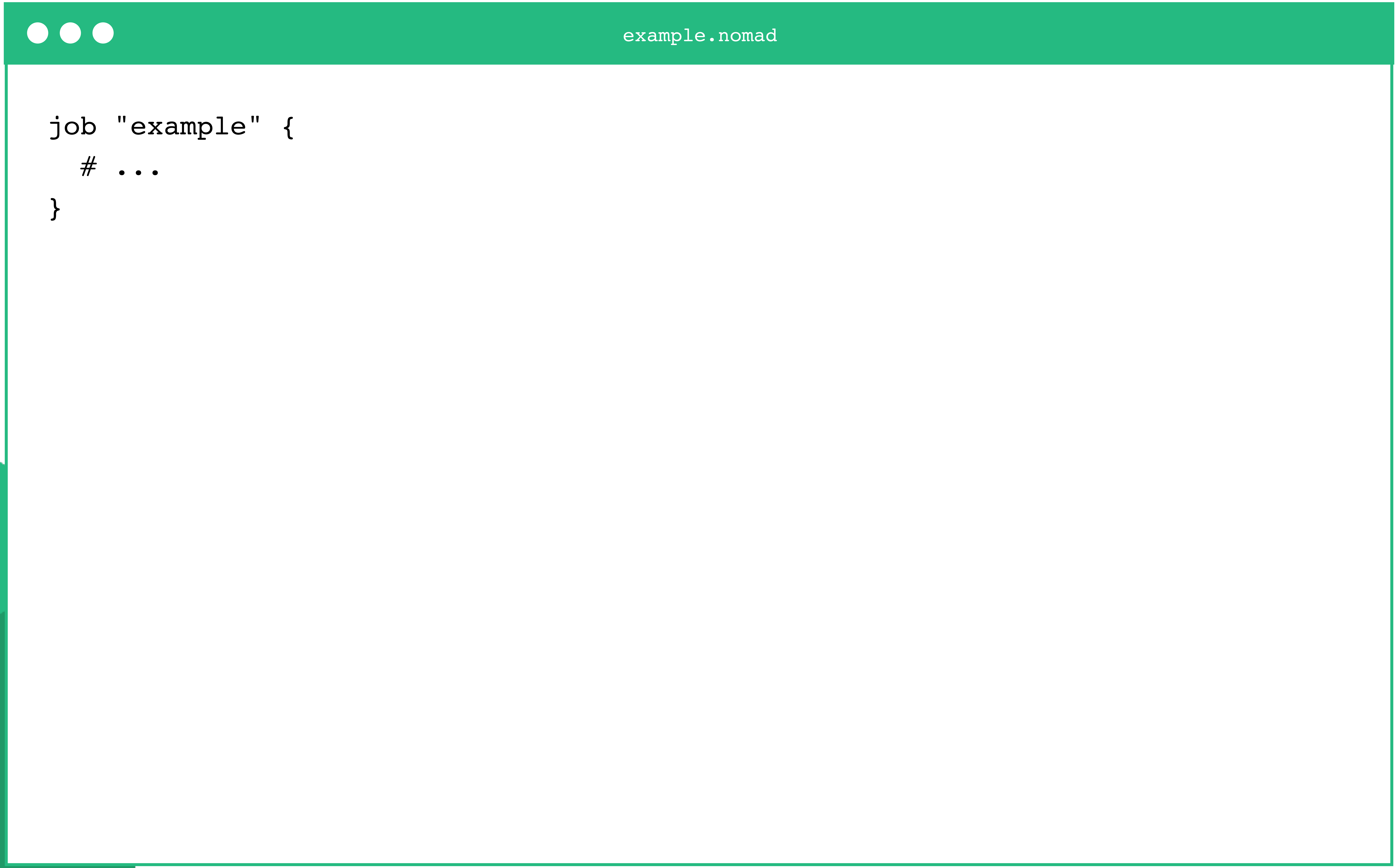
```
$ nomad init
Example job file written to example.nomad
```



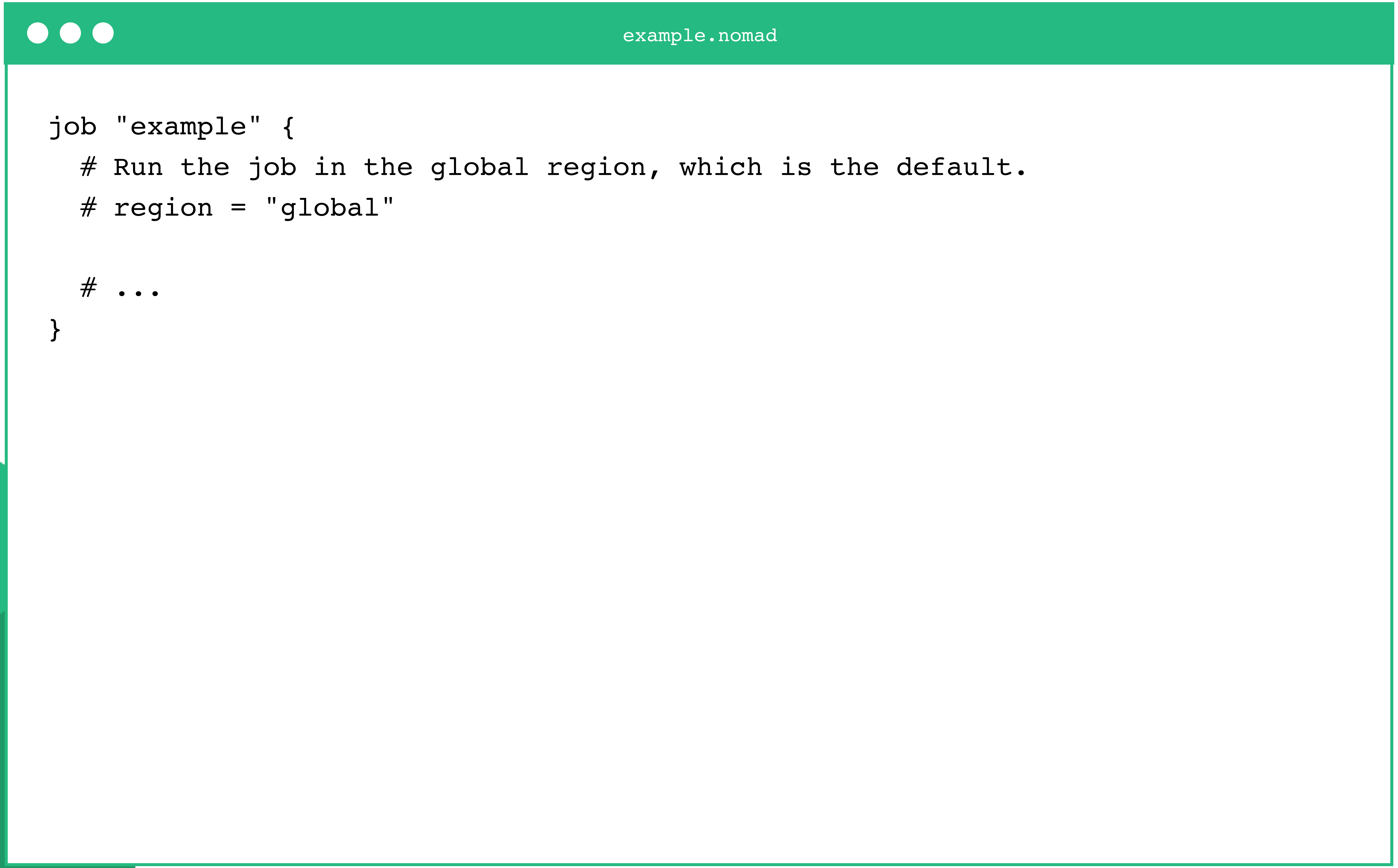
```
$ vi example.nomad
```

Nomad Jobs: Hierarchy

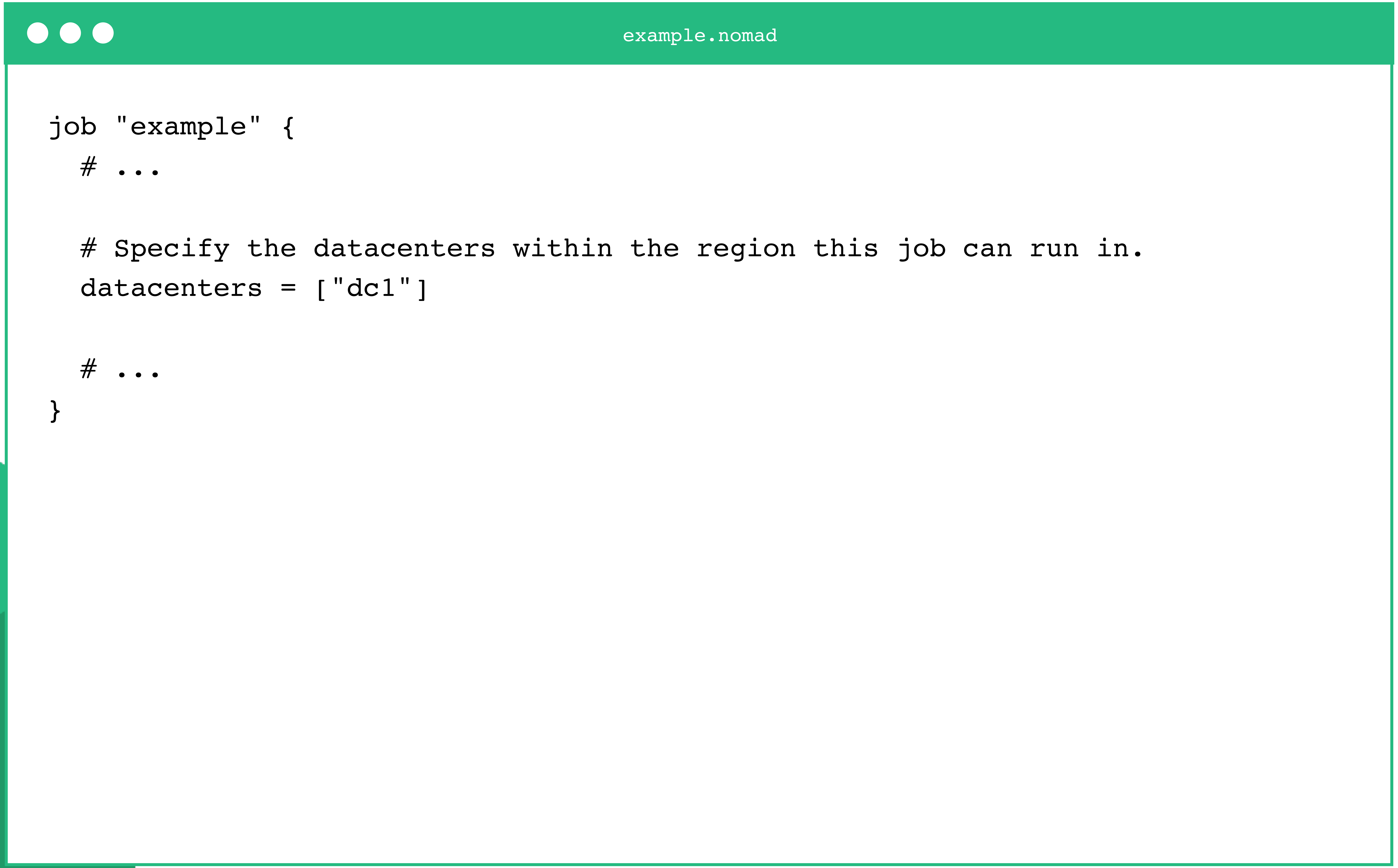
- Job
  - (Group)
    - Task
      - Resources
      - Constraint



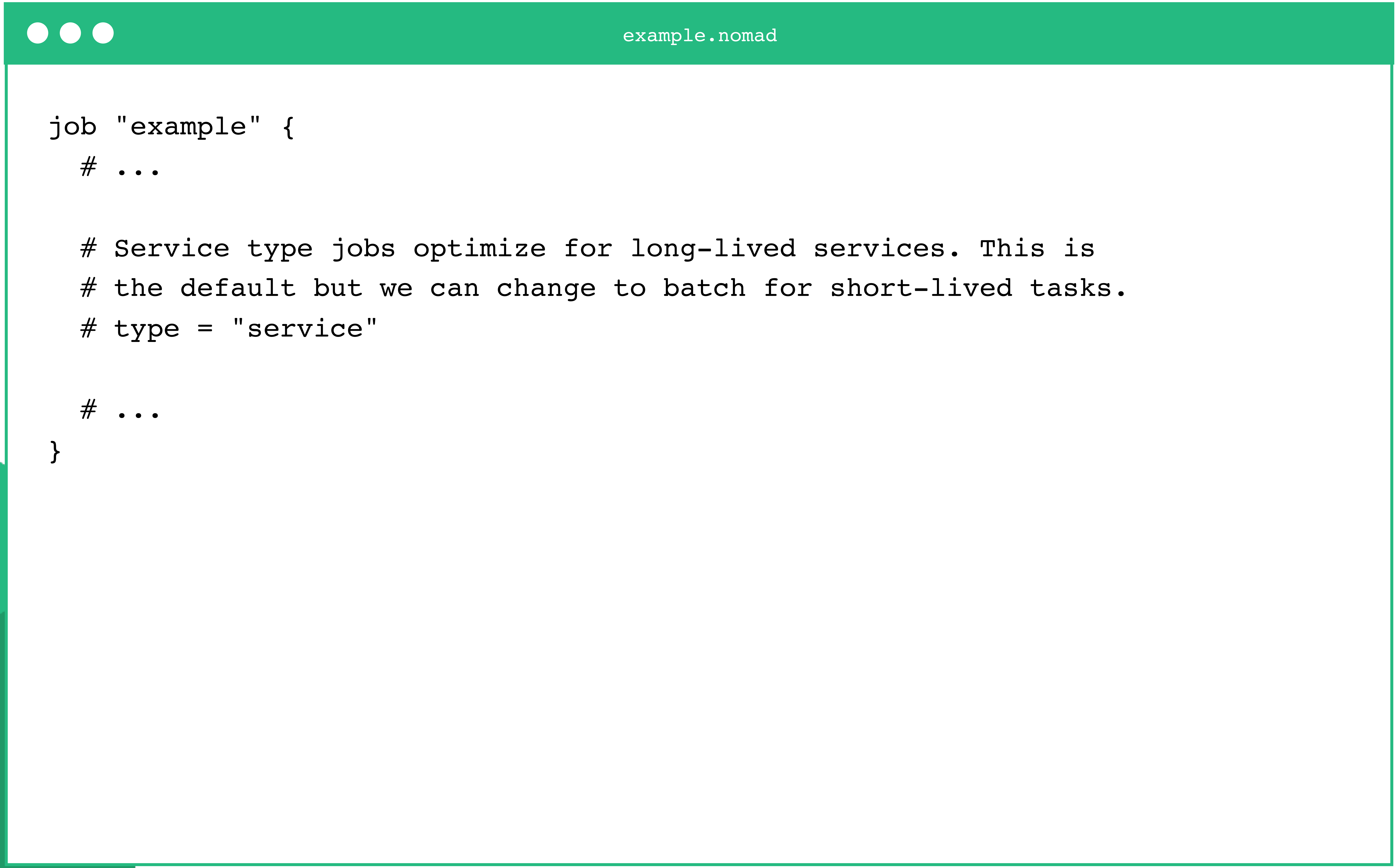
```
job "example" {
  # ...
}
```



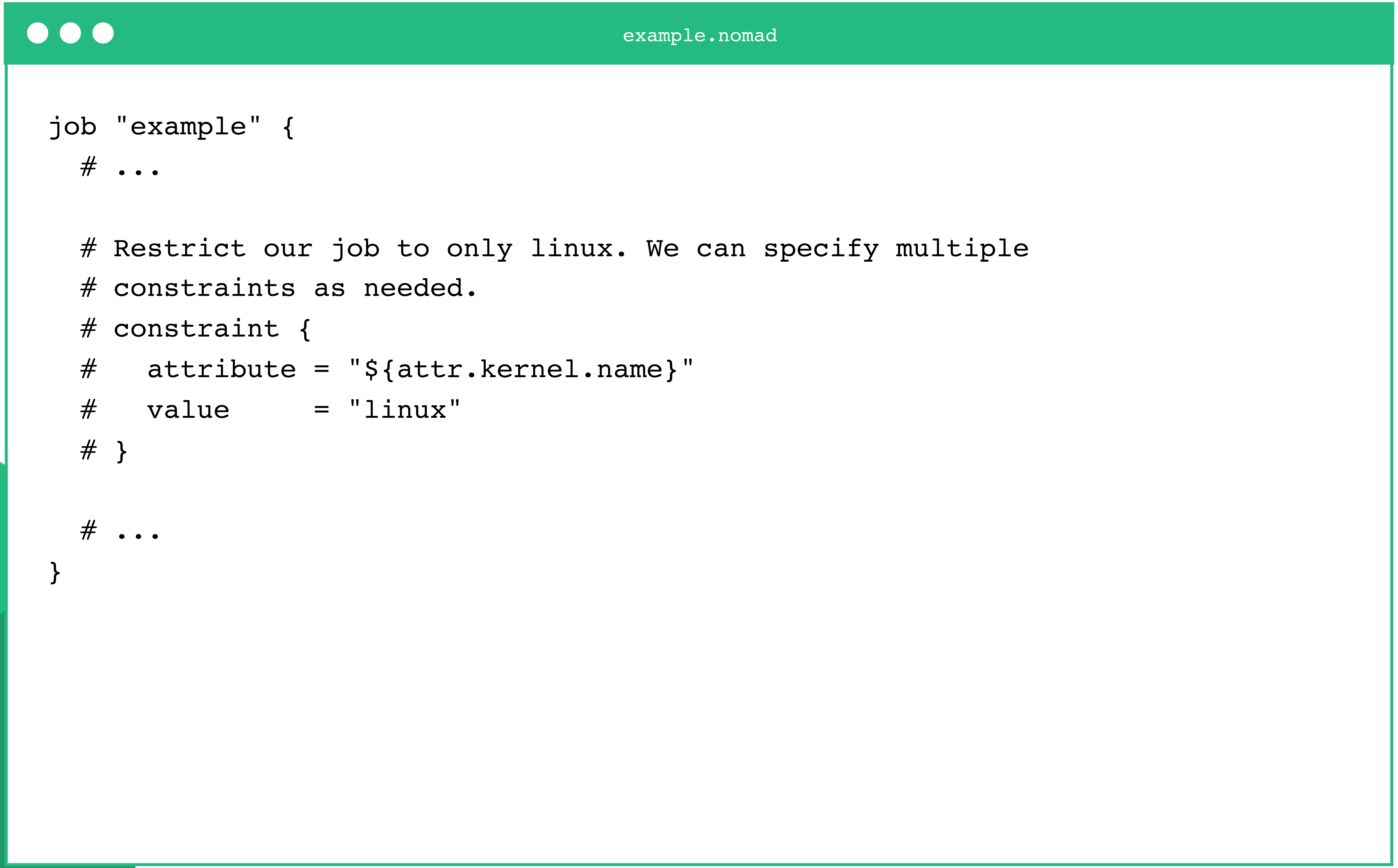
```
job "example" {
  # Run the job in the global region, which is the default.
  # region = "global"

  # ...
}
```



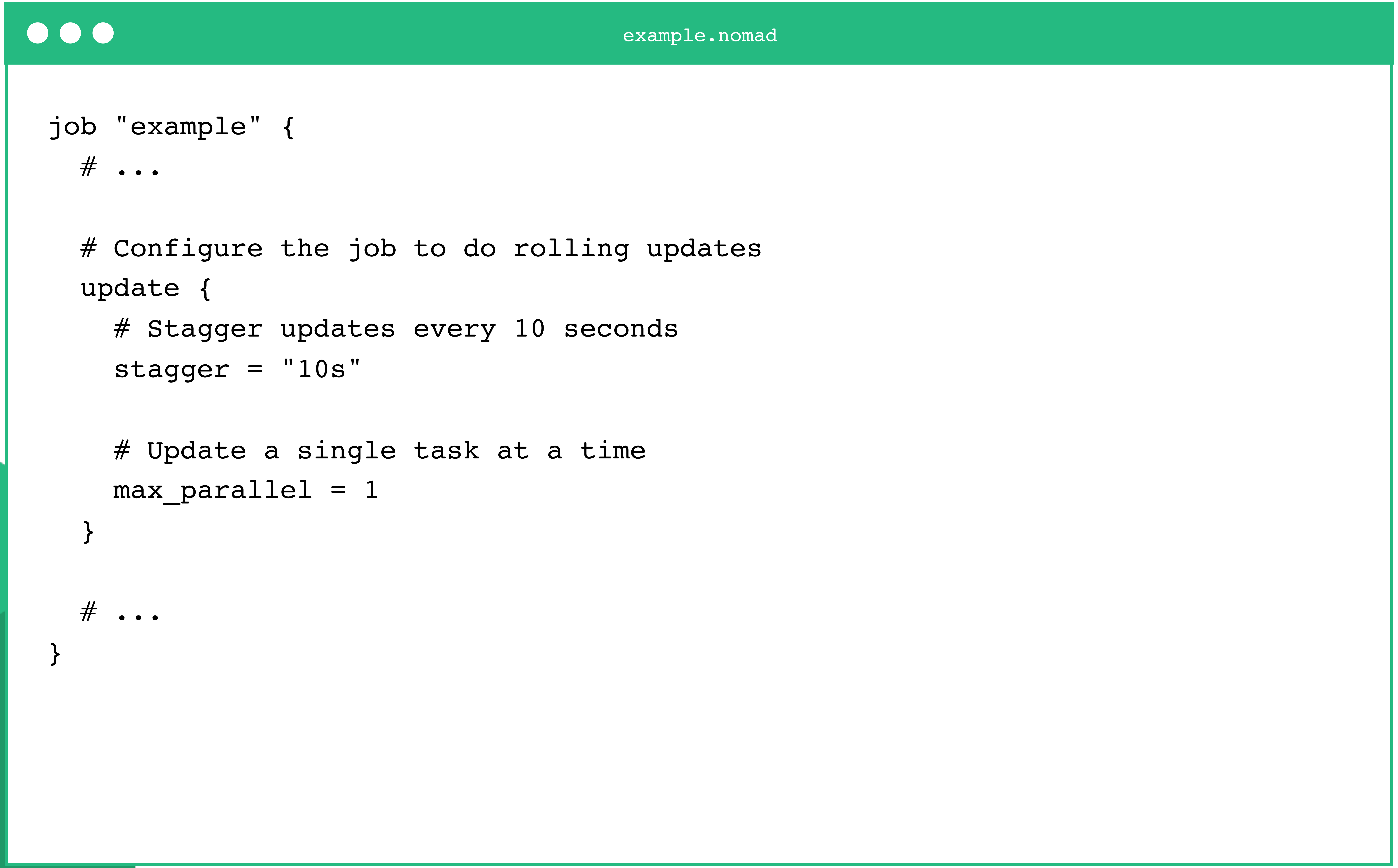
```
job "example" {
  # ...

  # Specify the datacenters within the region this job can run in.
  datacenters = ["dc1"]

  # ...
}
```



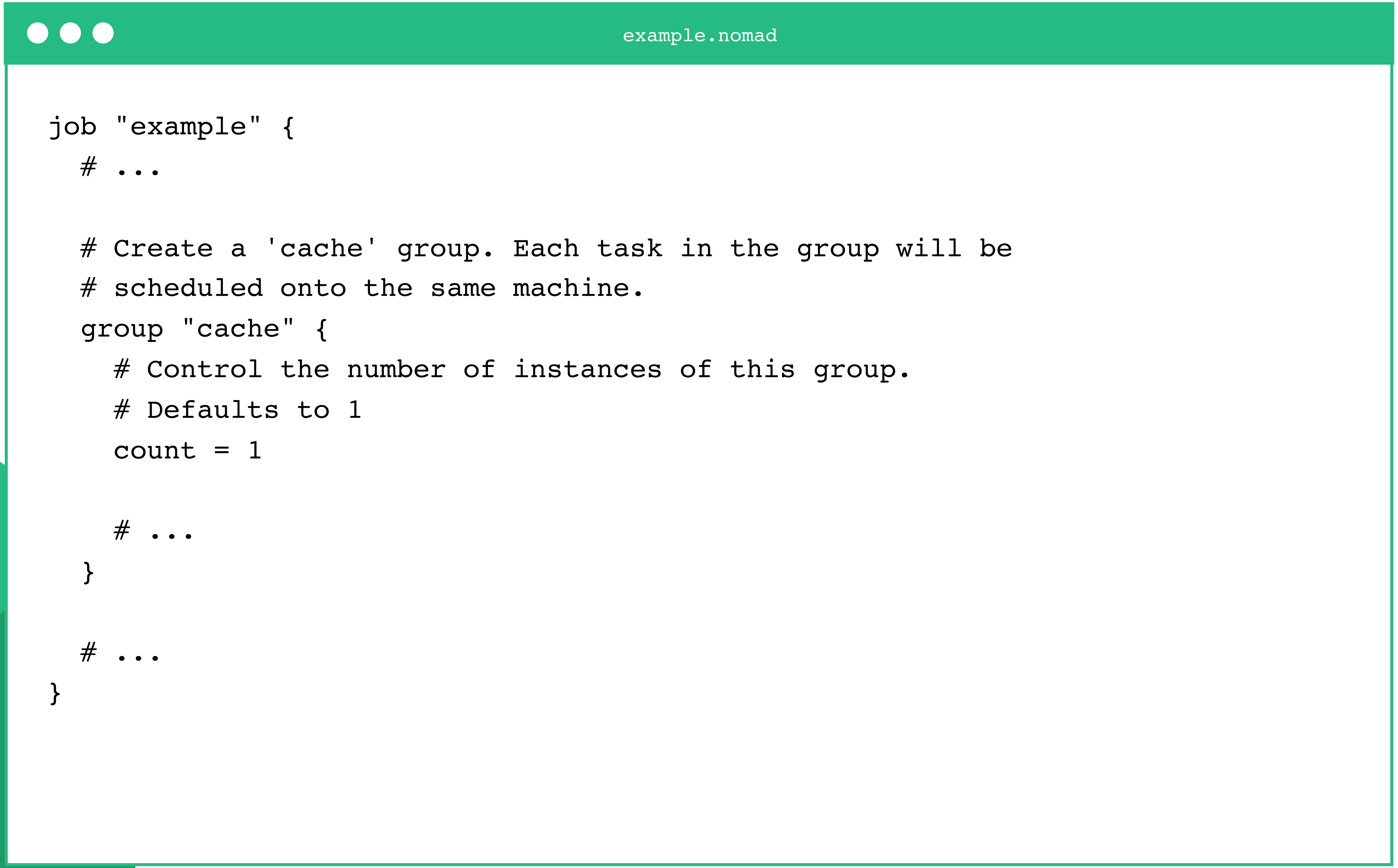
```
job "example" {
  # ...

  # Service type jobs optimize for long-lived services. This is
  # the default but we can change to batch for short-lived tasks.
  # type = "service"

  # ...
}
```



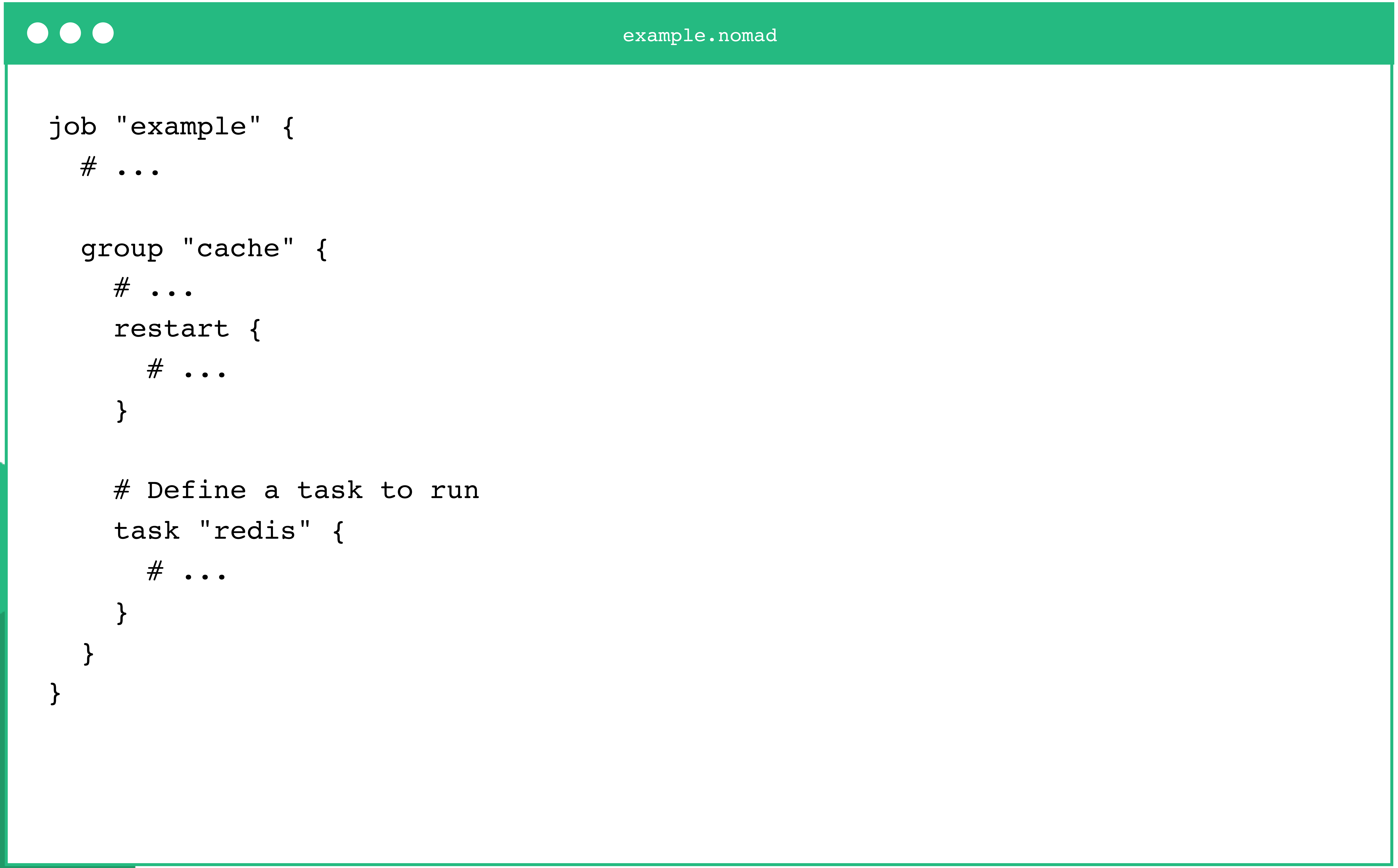
```
job "example" {
  # ...

  # Restrict our job to only linux. We can specify multiple
  # constraints as needed.
  # constraint {
  #   attribute = "${attr.kernel.name}"
  #   value     = "linux"
  # }

  # ...
}
```



```
job "example" {
  # ...

  # Configure the job to do rolling updates
  update {
    # Stagger updates every 10 seconds
    stagger = "10s"

    # Update a single task at a time
    max_parallel = 1
  }

  # ...
}
```

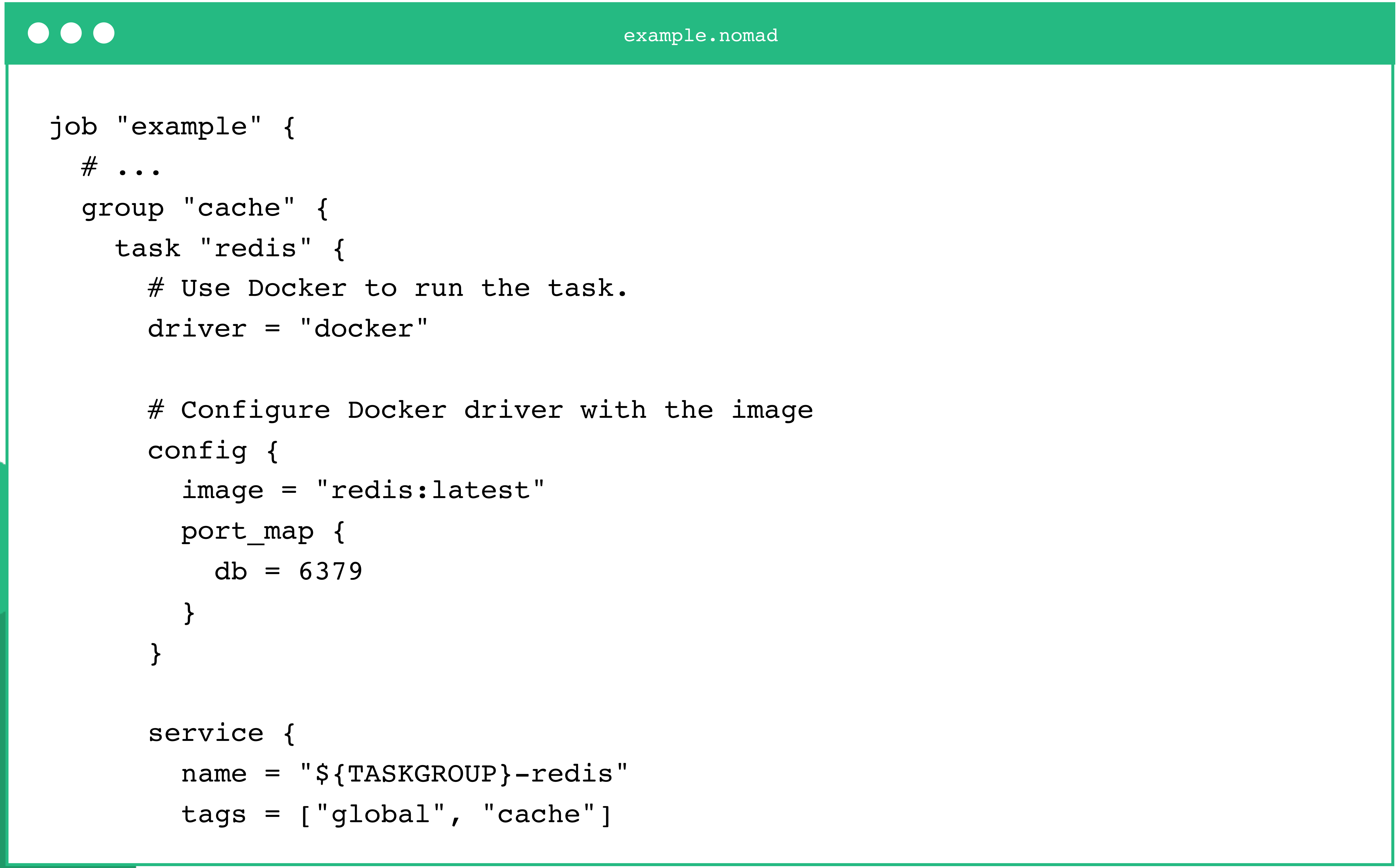


```
job "example" {
  # ...

  # Create a 'cache' group. Each task in the group will be
  # scheduled onto the same machine.
  group "cache" {
    # Control the number of instances of this group.
    # Defaults to 1
    count = 1

    # ...
  }

  # ...
}
```

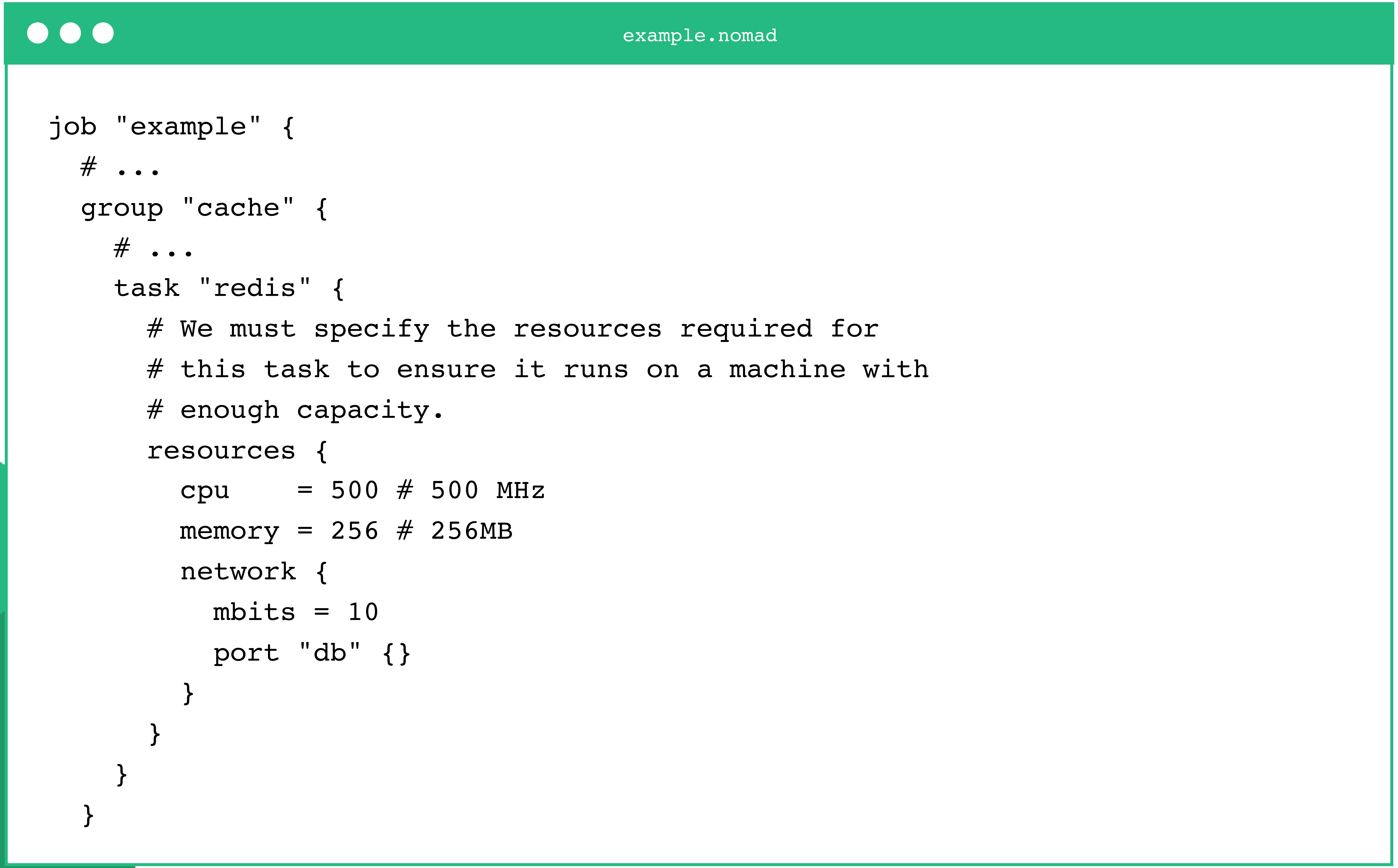


example.nomad

```
job "example" {
  # ...

  group "cache" {
    # ...
    restart {
      # ...
    }

    # Define a task to run
    task "redis" {
      # ...
    }
  }
}
```



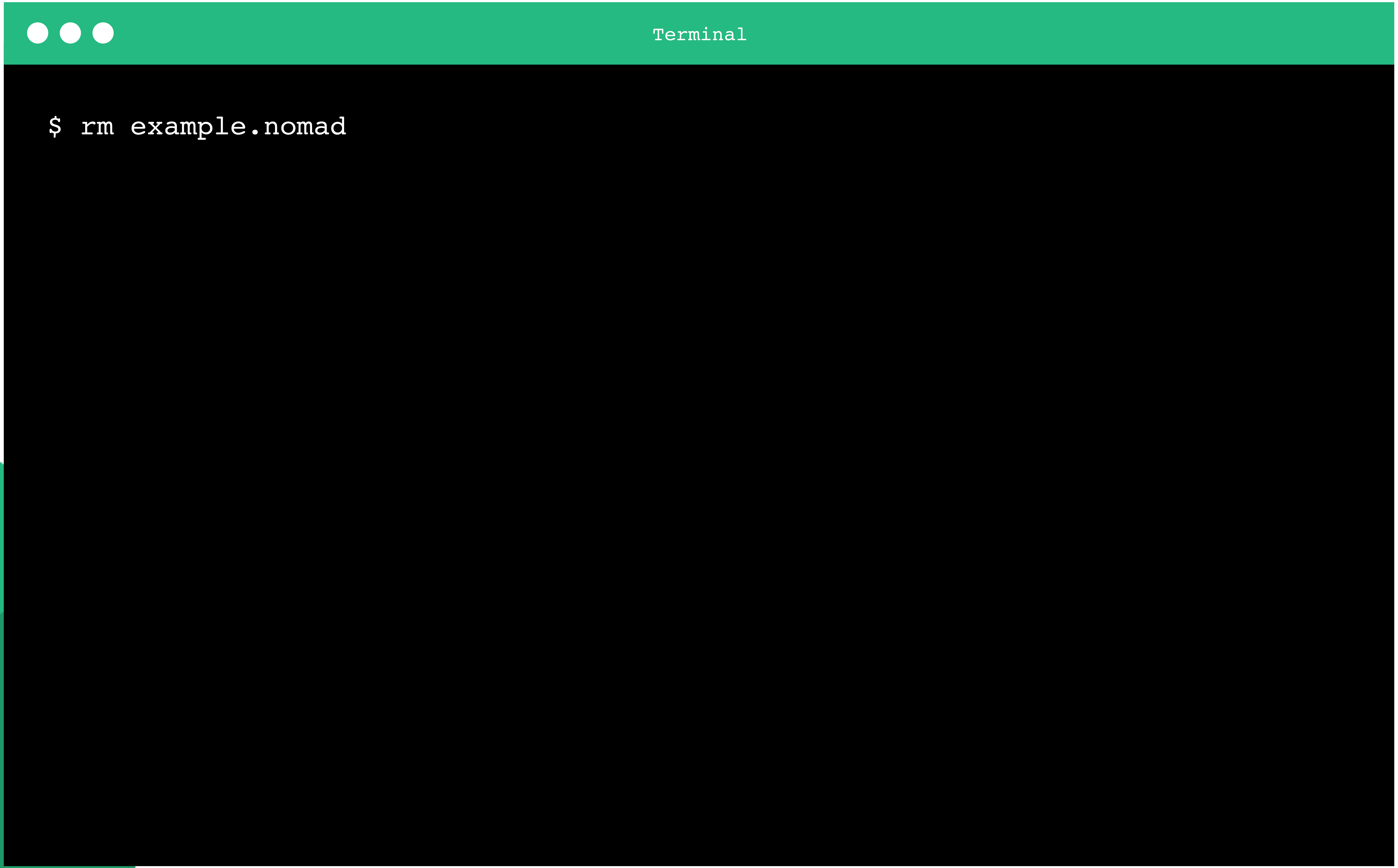
```
job "example" {
  # ...
  group "cache" {
    task "redis" {
      # Use Docker to run the task.
      driver = "docker"

      # Configure Docker driver with the image
      config {
        image = "redis:latest"
        port_map {
          db = 6379
        }
      }

      service {
        name = "${TASKGROUP}-redis"
        tags = ["global", "cache"]
```



```
job "example" {
  # ...
  group "cache" {
    # ...
    task "redis" {
      # We must specify the resources required for
      # this task to ensure it runs on a machine with
      # enough capacity.
      resources {
        cpu    = 500 # 500 MHz
        memory = 256 # 256MB
        network {
          mbits = 10
          port "db" {}
        }
      }
    }
  }
```

Exercise: Delete Example Job File

This job file is just an example, so delete it.



```
$ rm example.nomad
```

Exercise: Inspect Job File

There is already a job file on the system named `http-echo.nomad`.

Open and inspect this job file.

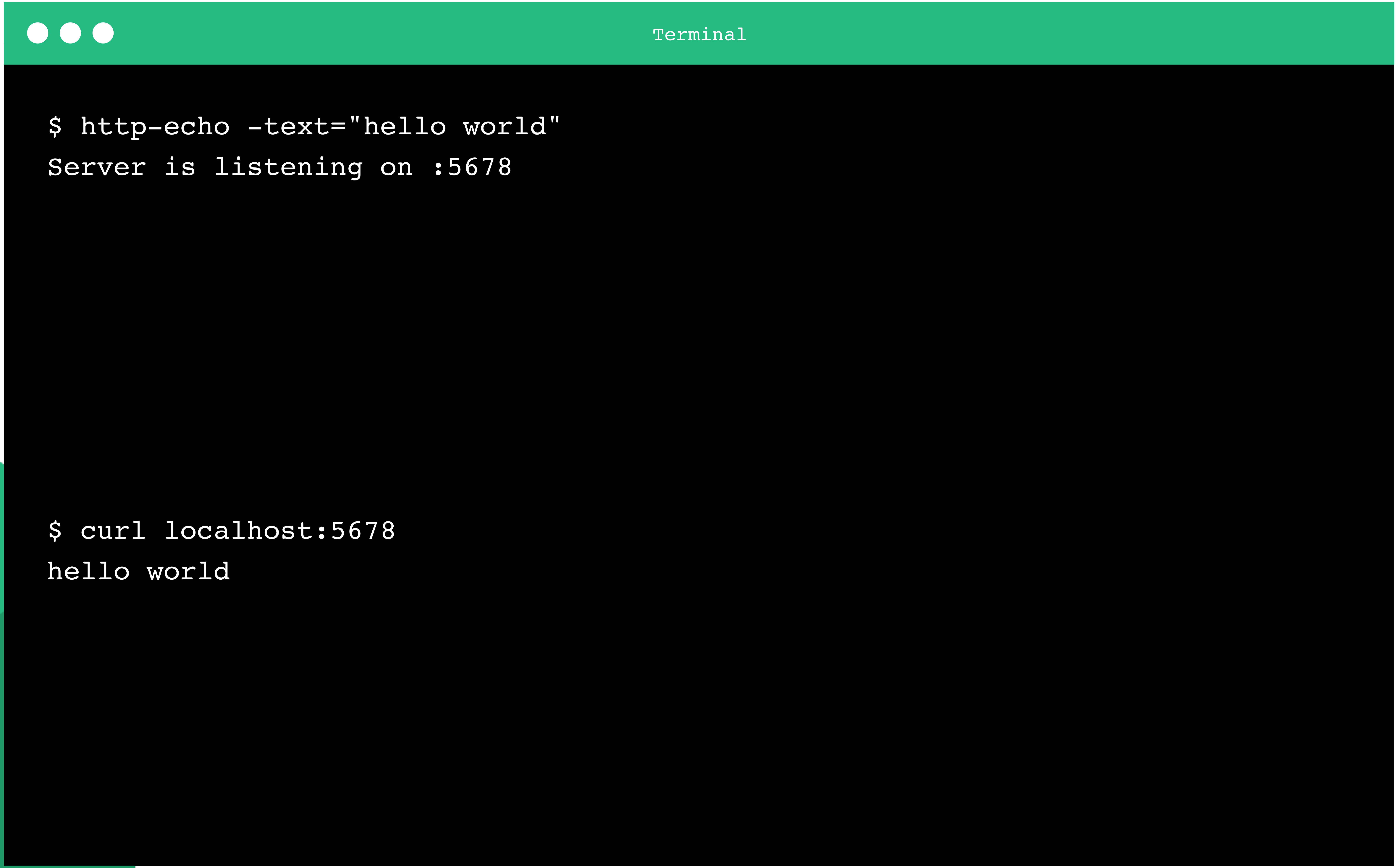
About: http-echo

`http-echo` is a small application that accepts text as a command-line flag and renders that text as an HTML webpage.

It accept `-listen` and `-text` flags.

Example invocation:

```
   http-echo -text="hello world"
```



```
$ http-echo -text="hello world"
Server is listening on :5678




$ curl localhost:5678
hello world
```

About: http-echo Docker Container

The `hashicorp/http-echo` Docker container packages the application in a distributable format.

It accepts the same arguments and flags as `http-echo`.

Example invocation:

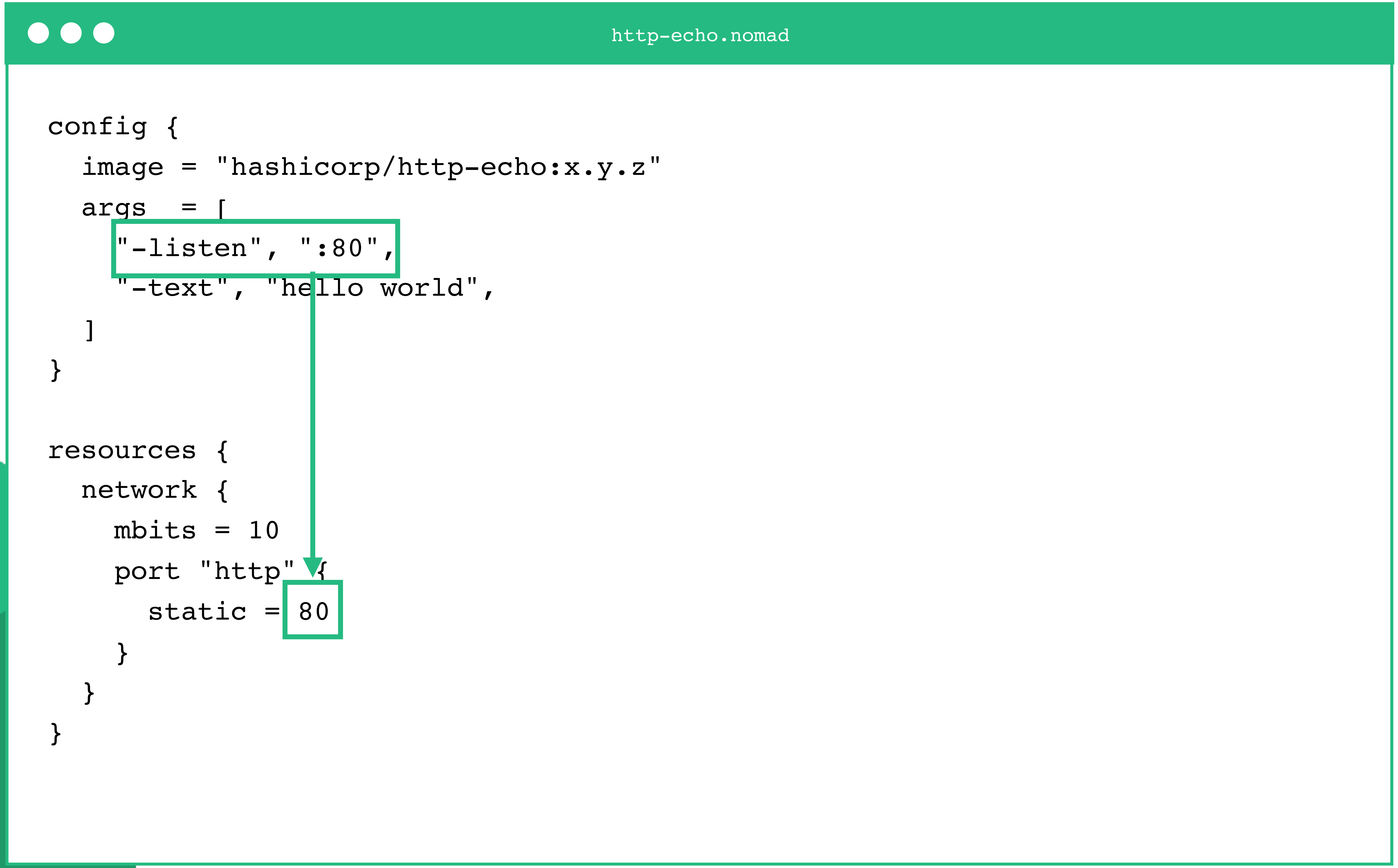
```
docker run hashicorp/http-echo -text="hello world"
```

About: http-echo Nomad Job

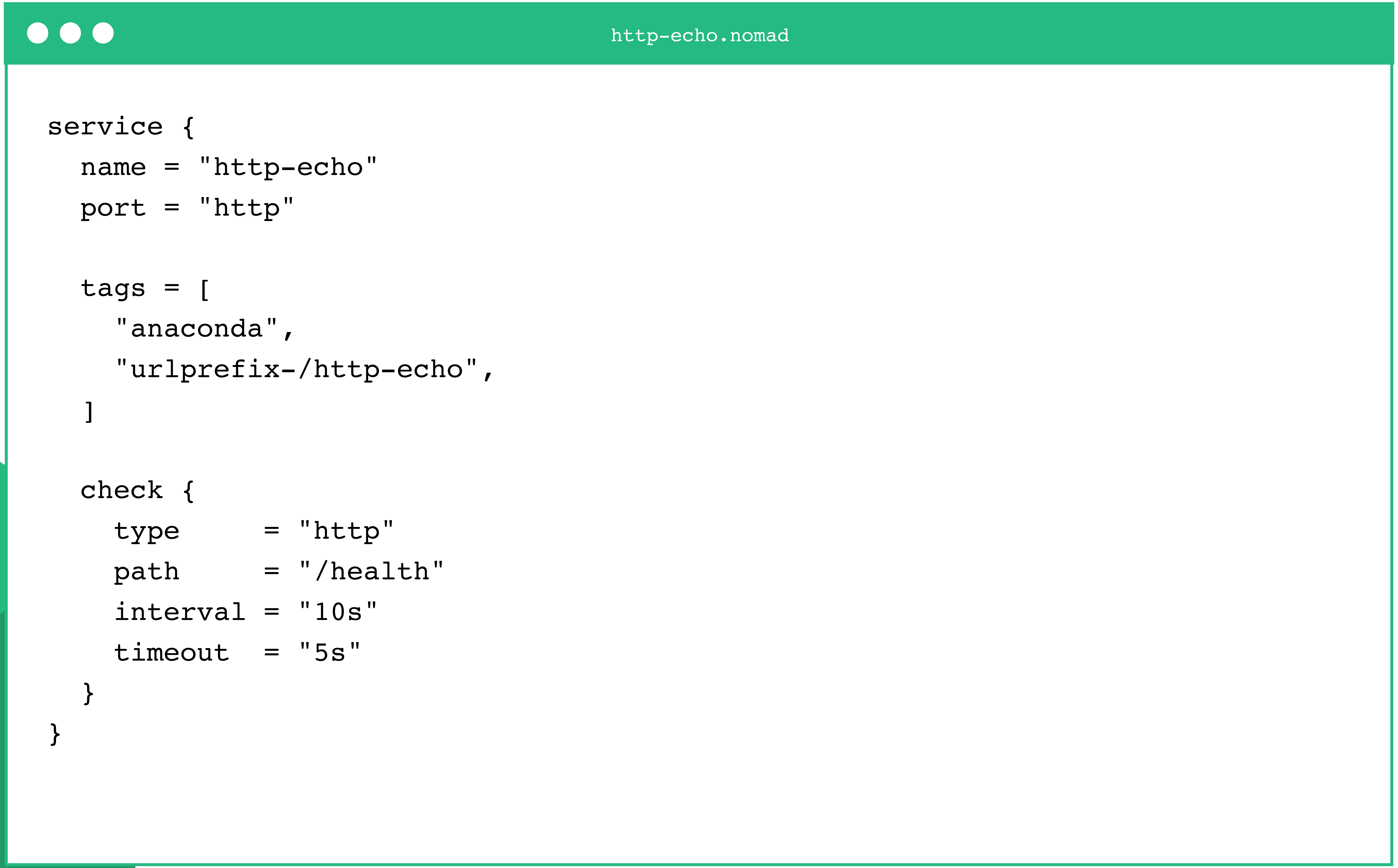
Requests Nomad to download and run a Docker container from the public docker registry (`hashicorp/http-echo`).

Configures the container to bind to port 80 on the host.

Registers the service for discovery with Consul with an integrated health check (more on this later).



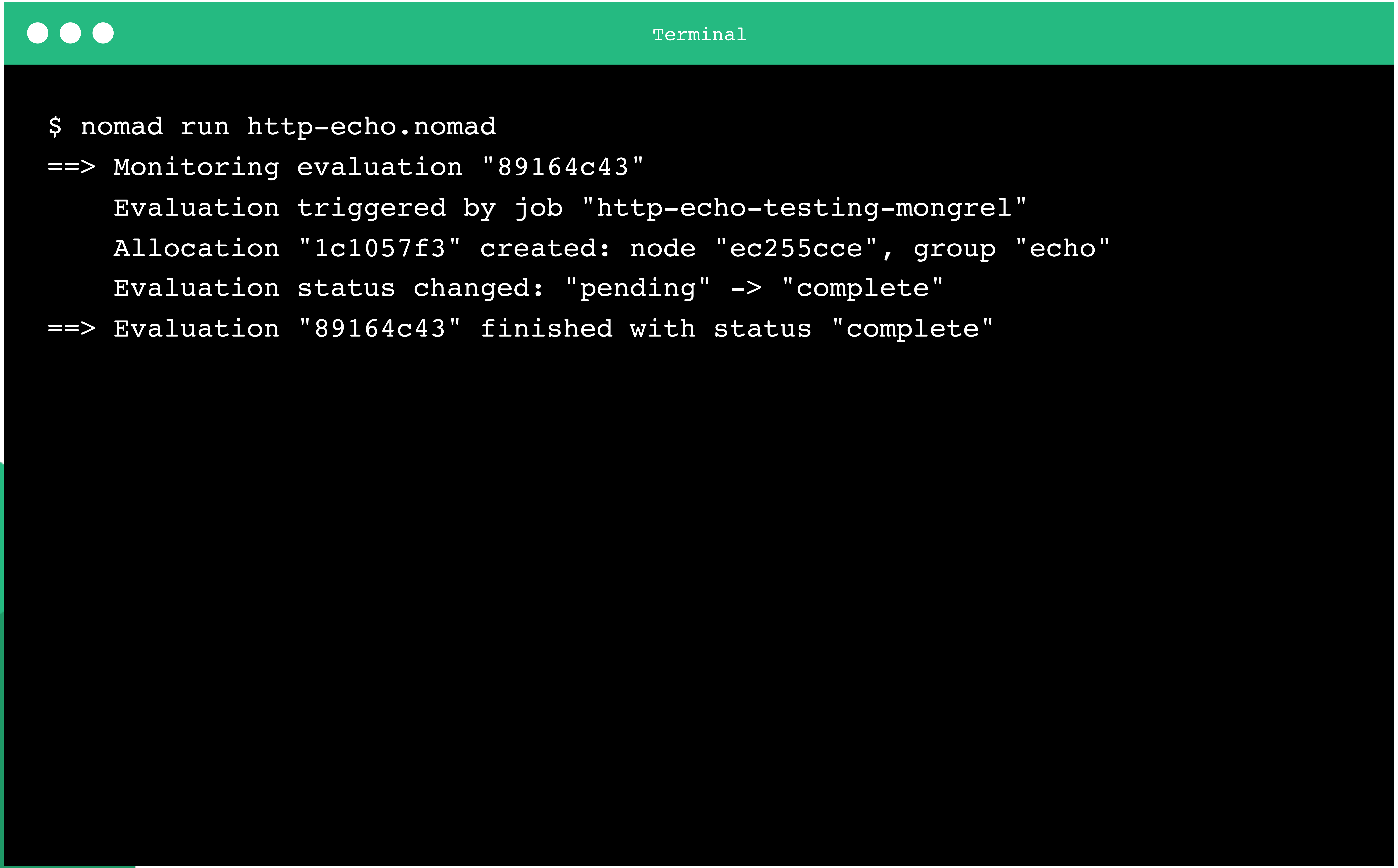
```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":80",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {
      static = 80
    }
  }
}
```



```
service {
  name = "http-echo"
  port = "http"

  tags = [
    "anaconda",
    "urlprefix-/http-echo",
  ]

  check {
    type     = "http"
    path     = "/health"
    interval = "10s"
    timeout  = "5s"
  }
}
```

Exercise: Run Nomad Job

Submit the `http-echo.nomad` job to the server for evaluation and scheduling.



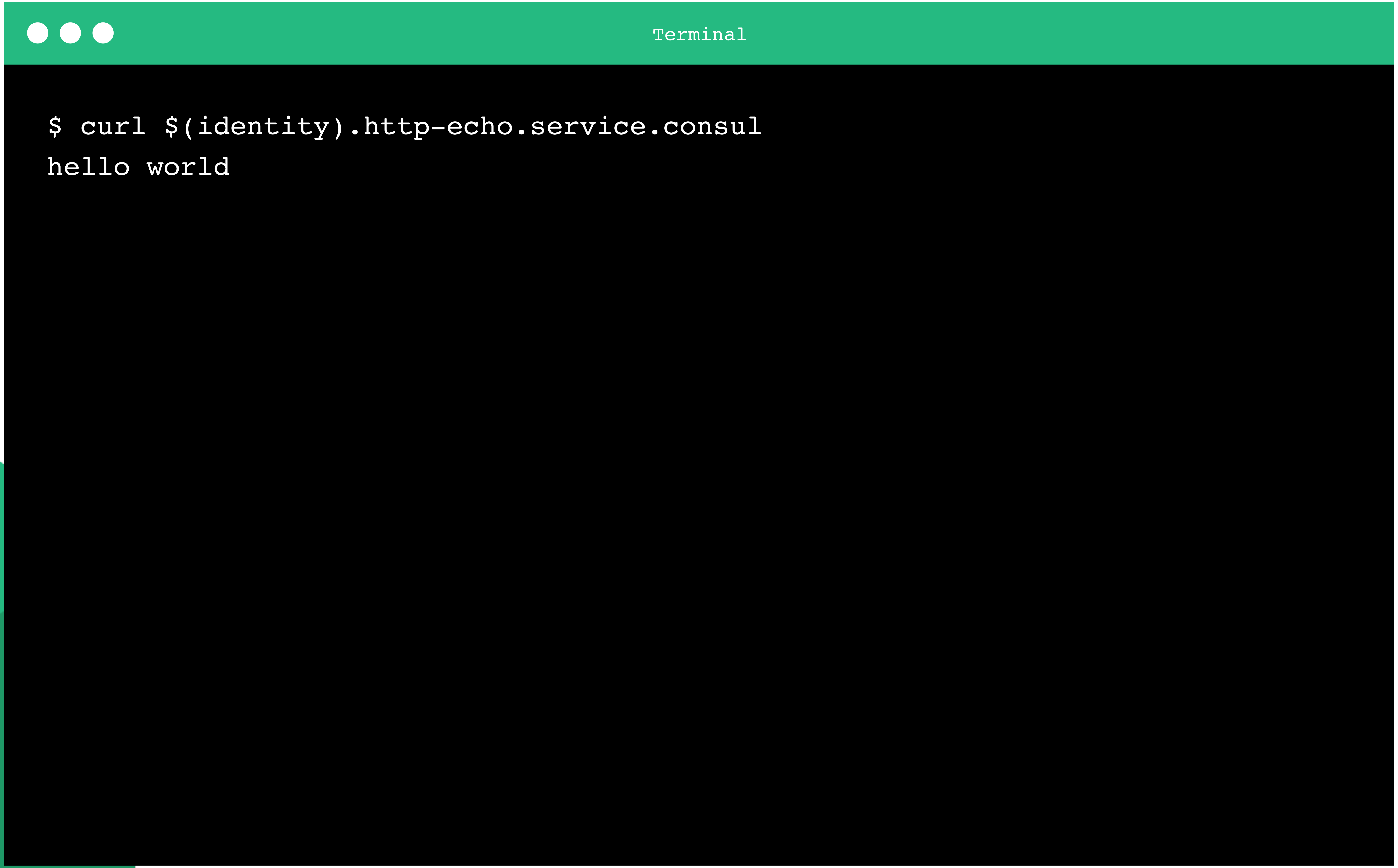
```
$ nomad run http-echo.nomad
==> Monitoring evaluation "89164c43"
    Evaluation triggered by job "http-echo-testing-mongrel"
    Allocation "1c1057f3" created: node "ec255cce", group "echo"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "89164c43" finished with status "complete"
```

Exercise: Contact Service

Your instance of `http-echo` was deployed on the cluster. It may have been scheduled on any host, so we need to use service discovery (Consul) to address it.

Use `curl` to query your instance

```
    $(identity).http-echo.service.consul
```



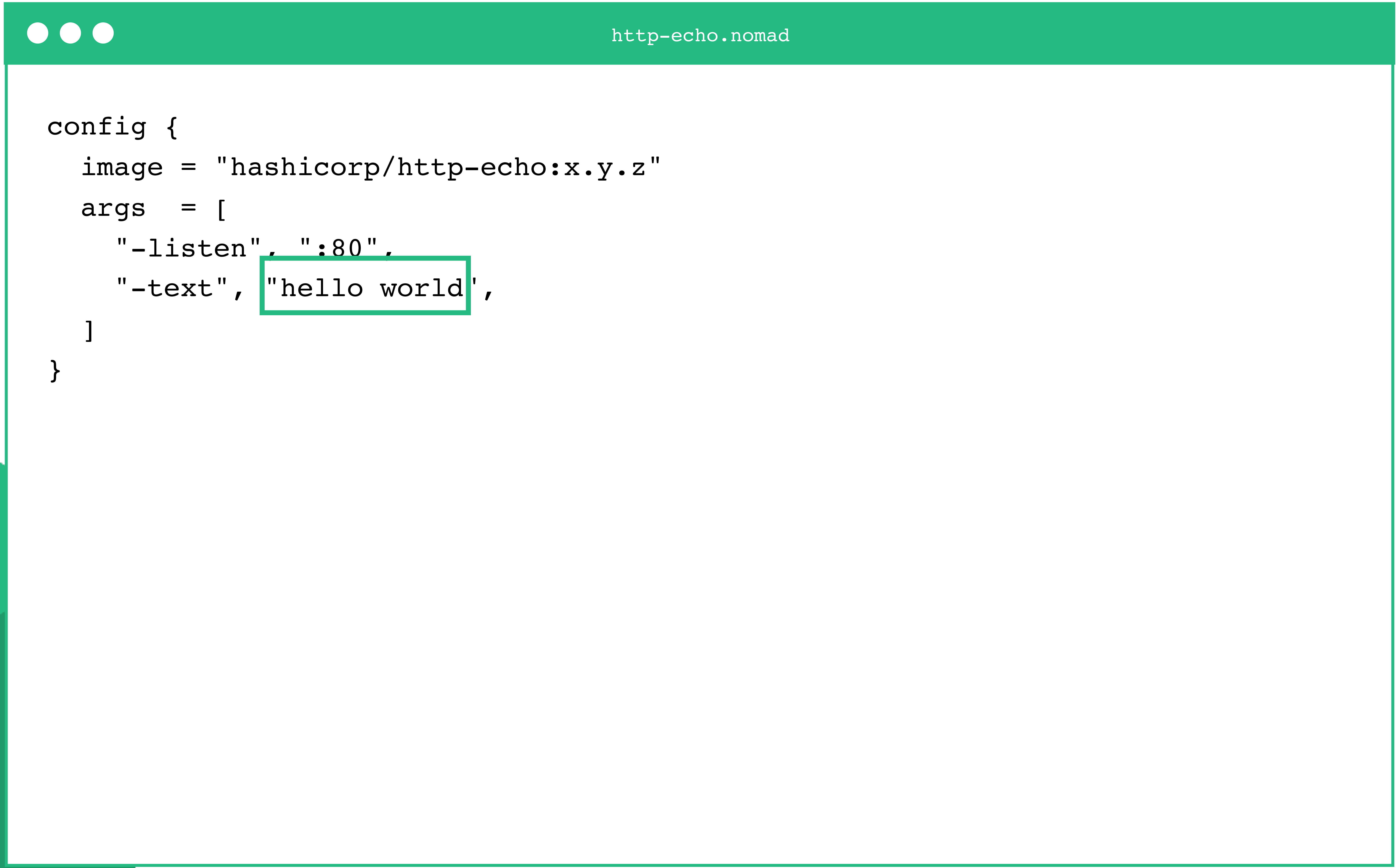
```
$ curl $(identity).http-echo.service.consul
hello world
```

Exercise: Change Job

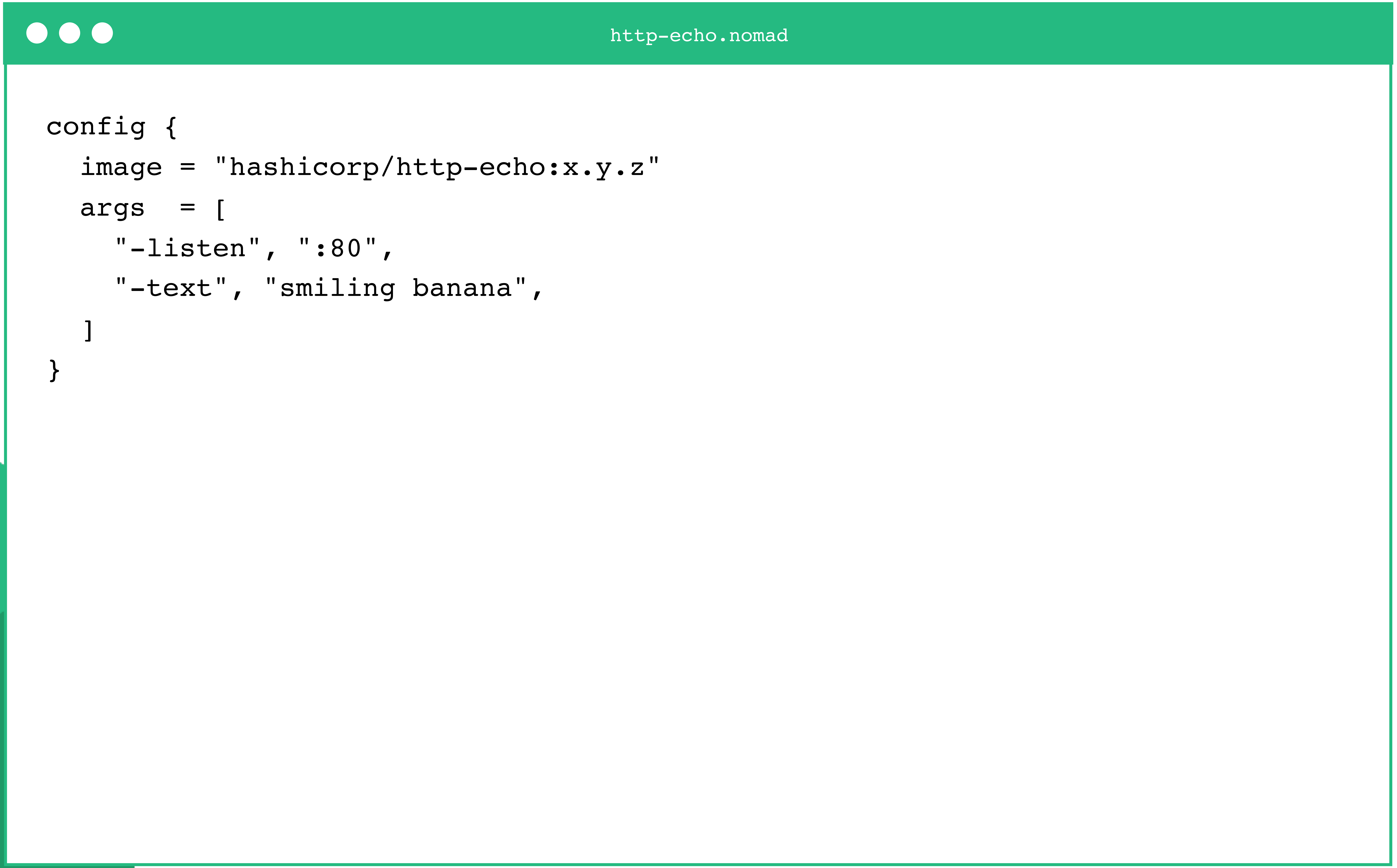
Modify the job file to render your name (or anything you choose) instead of "hello world".

Resubmit the modified job to Nomad.

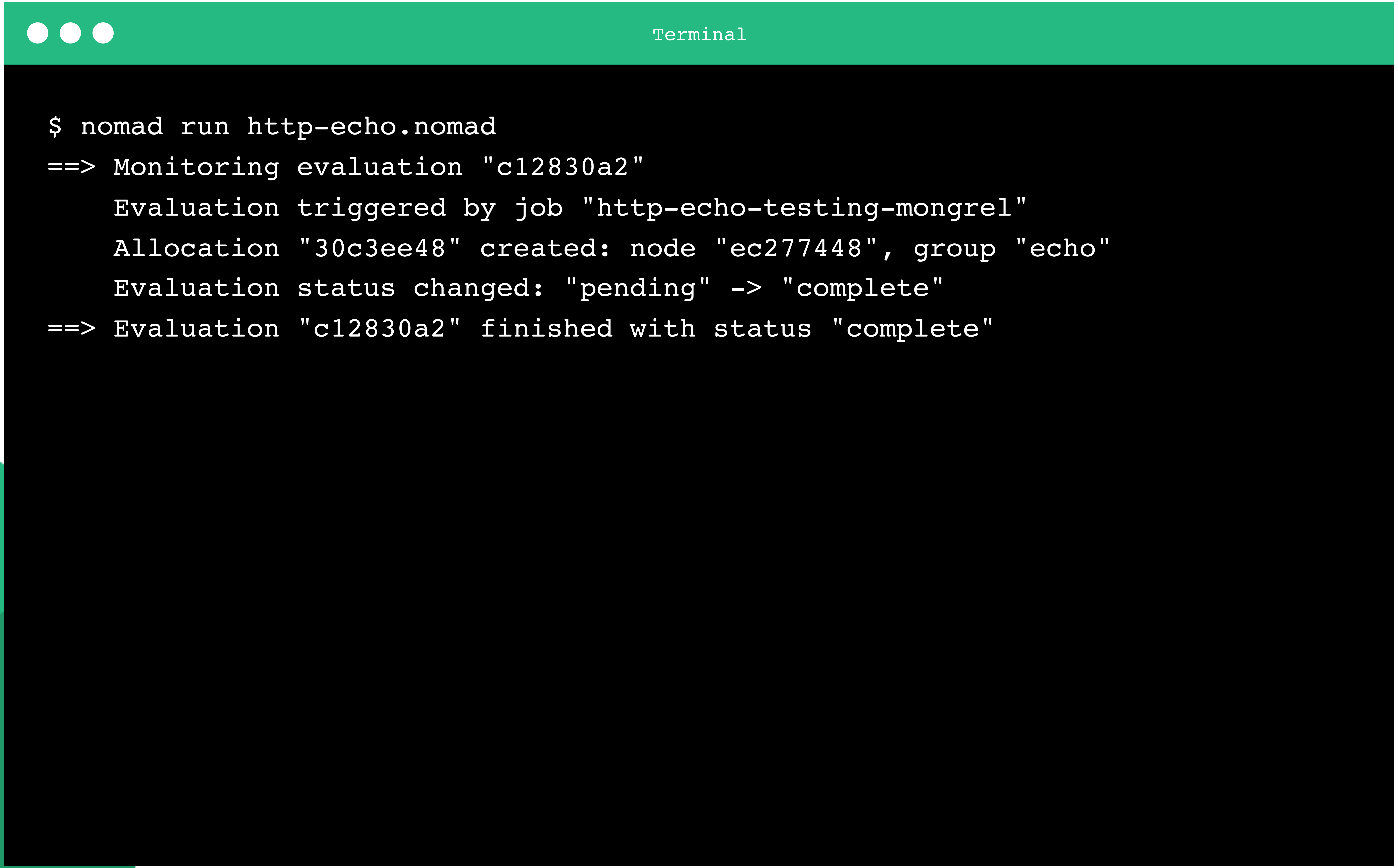
Use `curl` to query the updated instance.



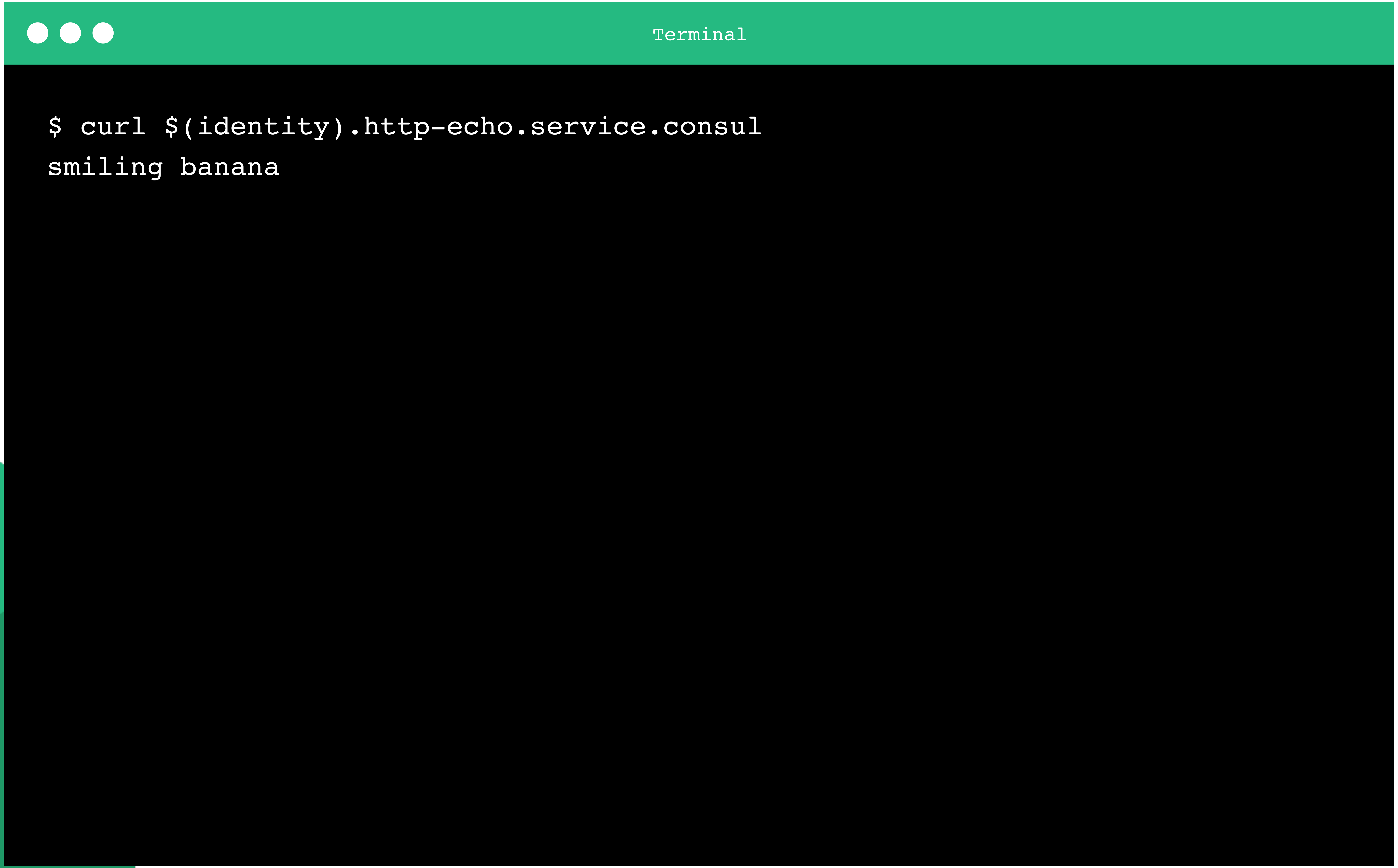
```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":80",
    "-text", "hello world",
  ]
}
```



```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":80",
    "-text", "smiling banana",
  ]
}
```



```
$ nomad run http-echo.nomad
==> Monitoring evaluation "c12830a2"
    Evaluation triggered by job "http-echo-testing-mongrel"
    Allocation "30c3ee48" created: node "ec277448", group "echo"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "c12830a2" finished with status "complete"
```



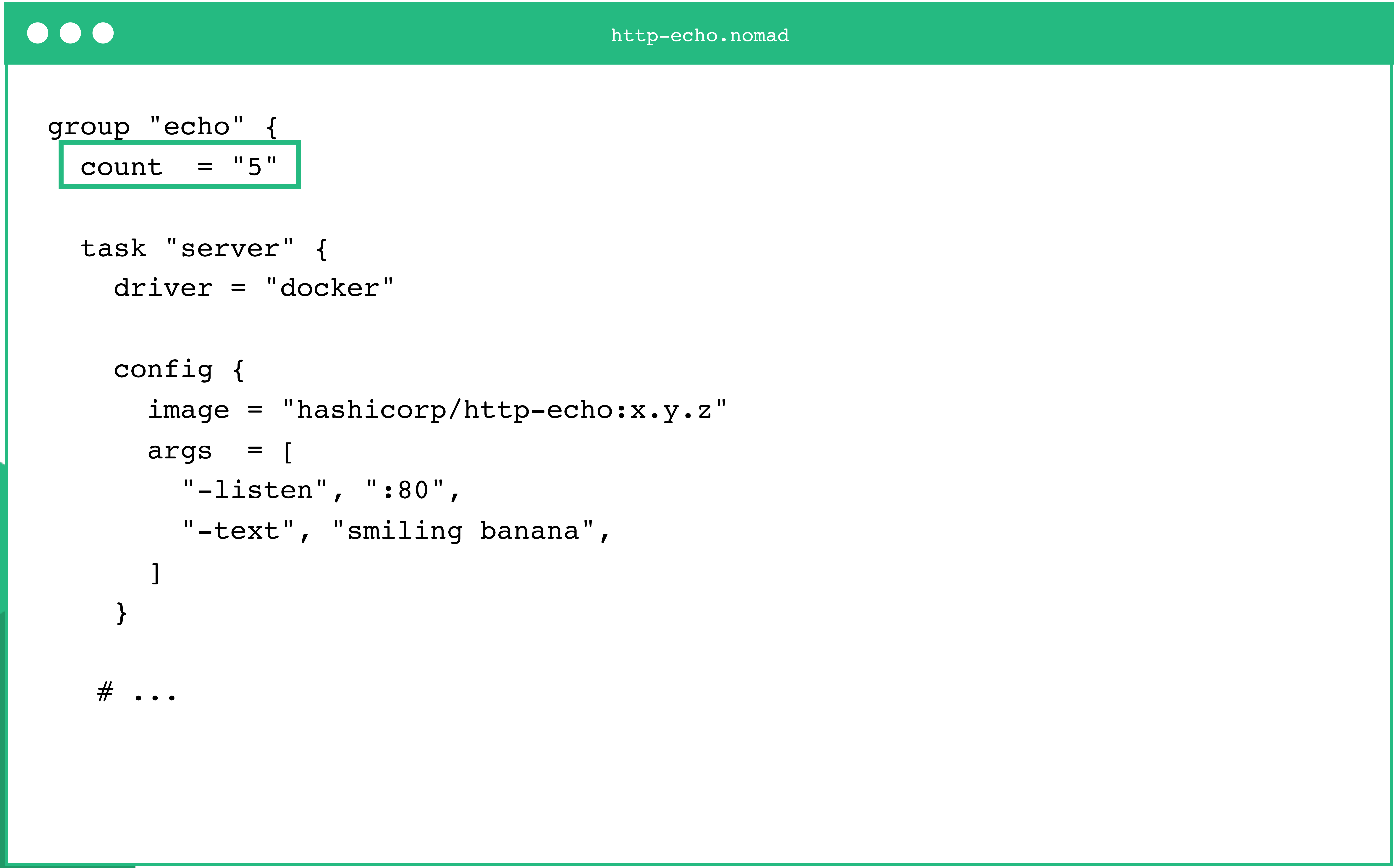
```
$ curl $(identity).http-echo.service.consul
smiling banana
```

Exercise: Scale `http-echo` Job

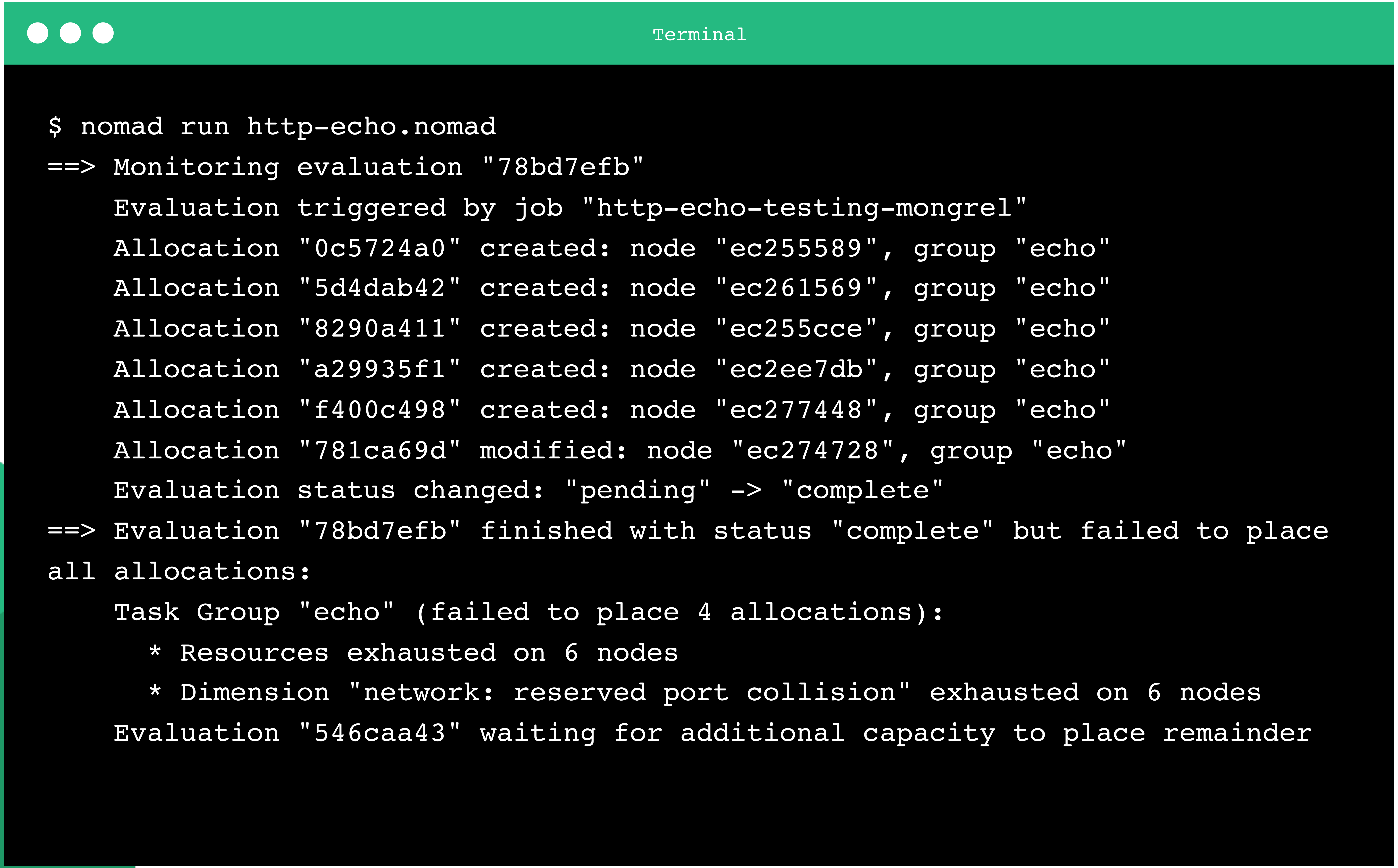
Increase the `count` attribute on the group to "5" to run five instances of our application.

Resubmit this job for evaluation.

HINT: You may need to add a `count` attribute or look online at the documentation.



```
group "echo" {
  count   = "5"

  task "server" {
    driver = "docker"

    config {
      image = "hashicorp/http-echo:x.y.z"
      args  = [
        "-listen", ":80",
        "-text", "smiling banana",
      ]
    }

   # ...
```

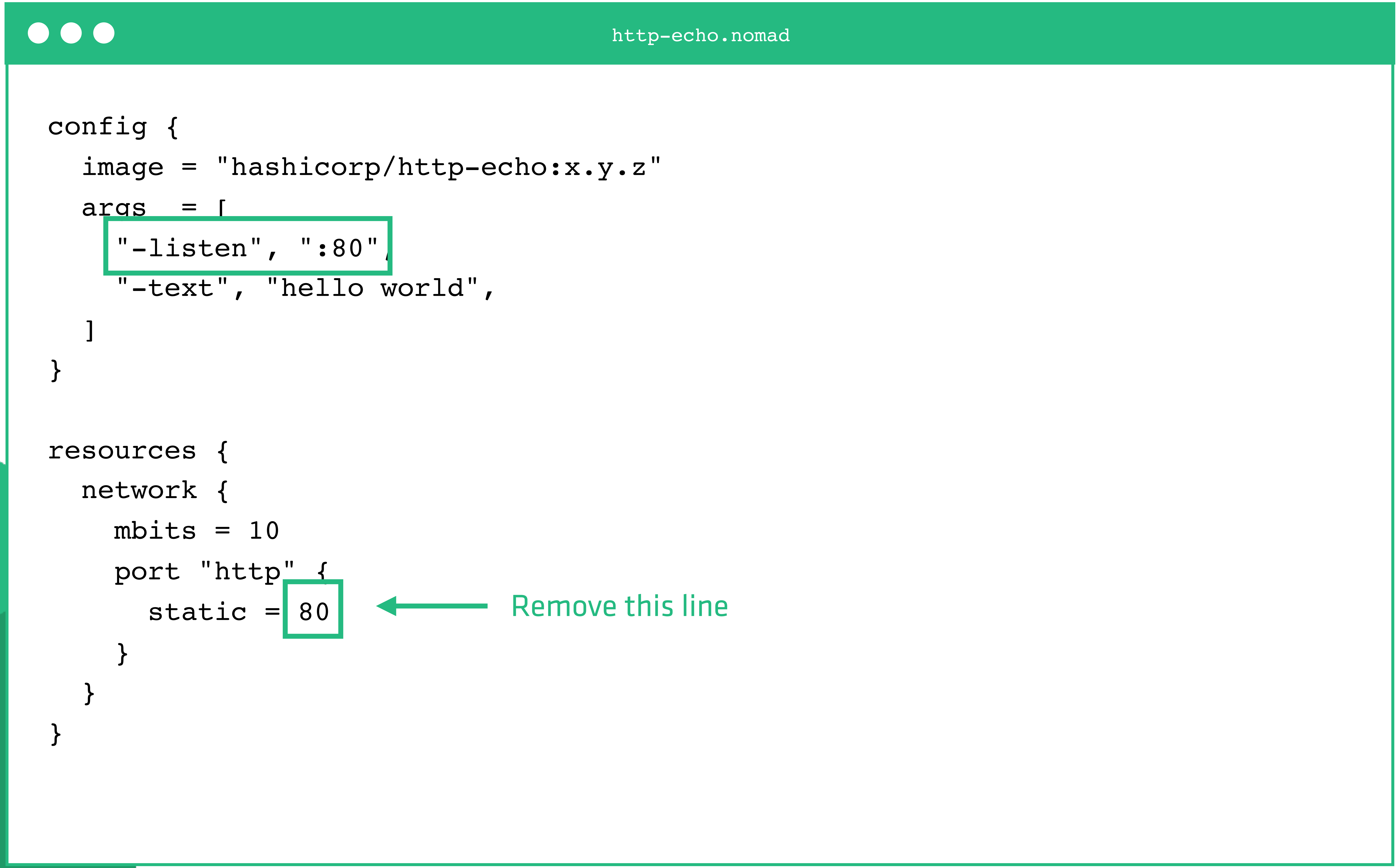


```
$ nomad run http-echo.nomad
==> Monitoring evaluation "78bd7efb"
    Evaluation triggered by job "http-echo-testing-mongrel"
    Allocation "0c5724a0" created: node "ec255589", group "echo"
    Allocation "5d4dab42" created: node "ec261569", group "echo"
    Allocation "8290a411" created: node "ec255cce", group "echo"
    Allocation "a29935f1" created: node "ec2ee7db", group "echo"
    Allocation "f400c498" created: node "ec277448", group "echo"
    Allocation "781ca69d" modified: node "ec274728", group "echo"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "78bd7efb" finished with status "complete" but failed to place
all allocations:
    Task Group "echo" (failed to place 4 allocations):
      * Resources exhausted on 6 nodes
      * Dimension "network: reserved port collision" exhausted on 6 nodes
    Evaluation "546caa43" waiting for additional capacity to place remainder
```

About: Static Ports

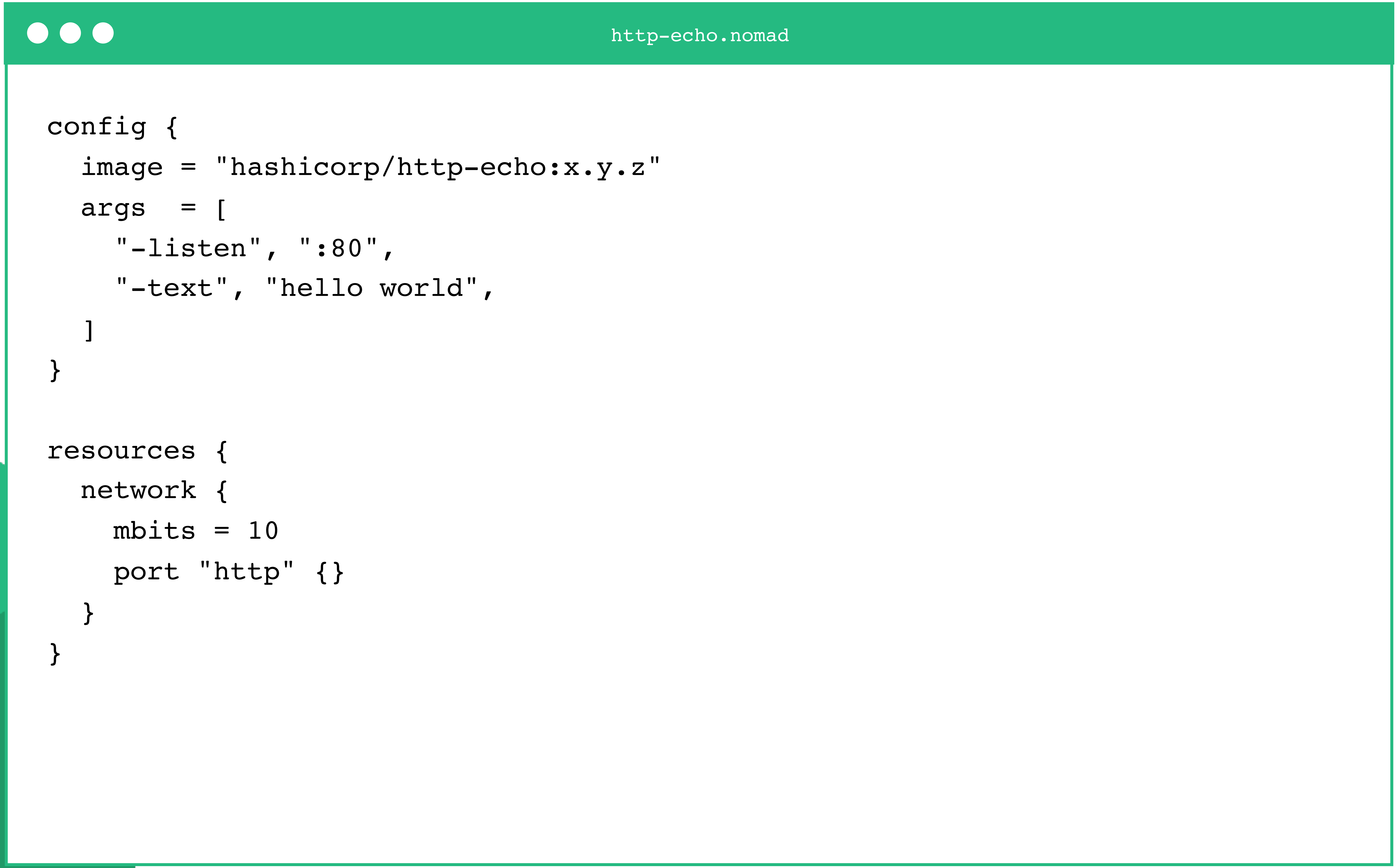
Static ports are generally a bad idea in scheduled environments as they restrict exactly one instance of the job to running per host.

Letting Nomad choose dynamic ports allows for better scale.

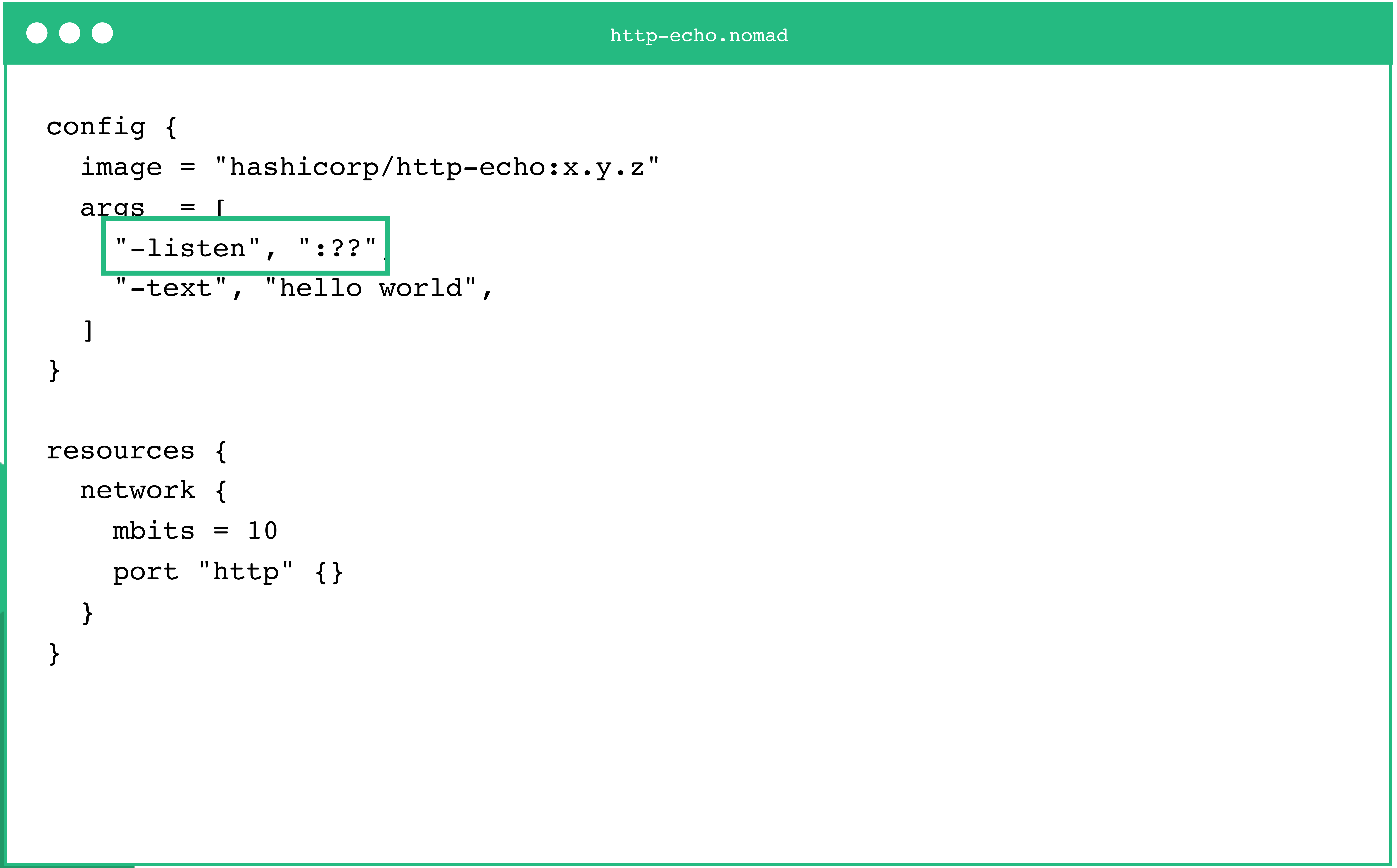


```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":80",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {
      static = 80
    }
  }
}
```

Remove this line



```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":80",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {}
  }
}
```

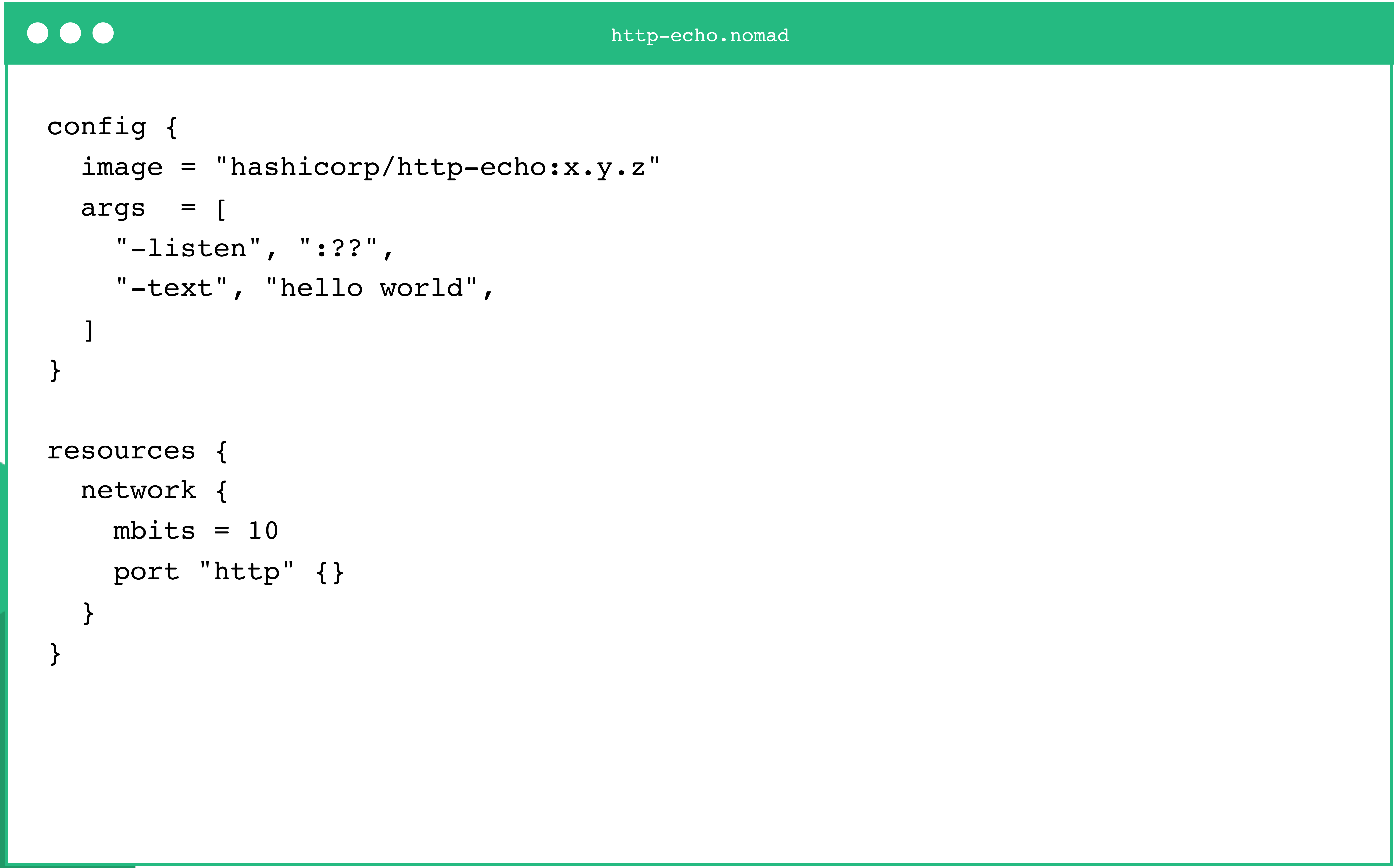


```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":??",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {}
  }
}
```

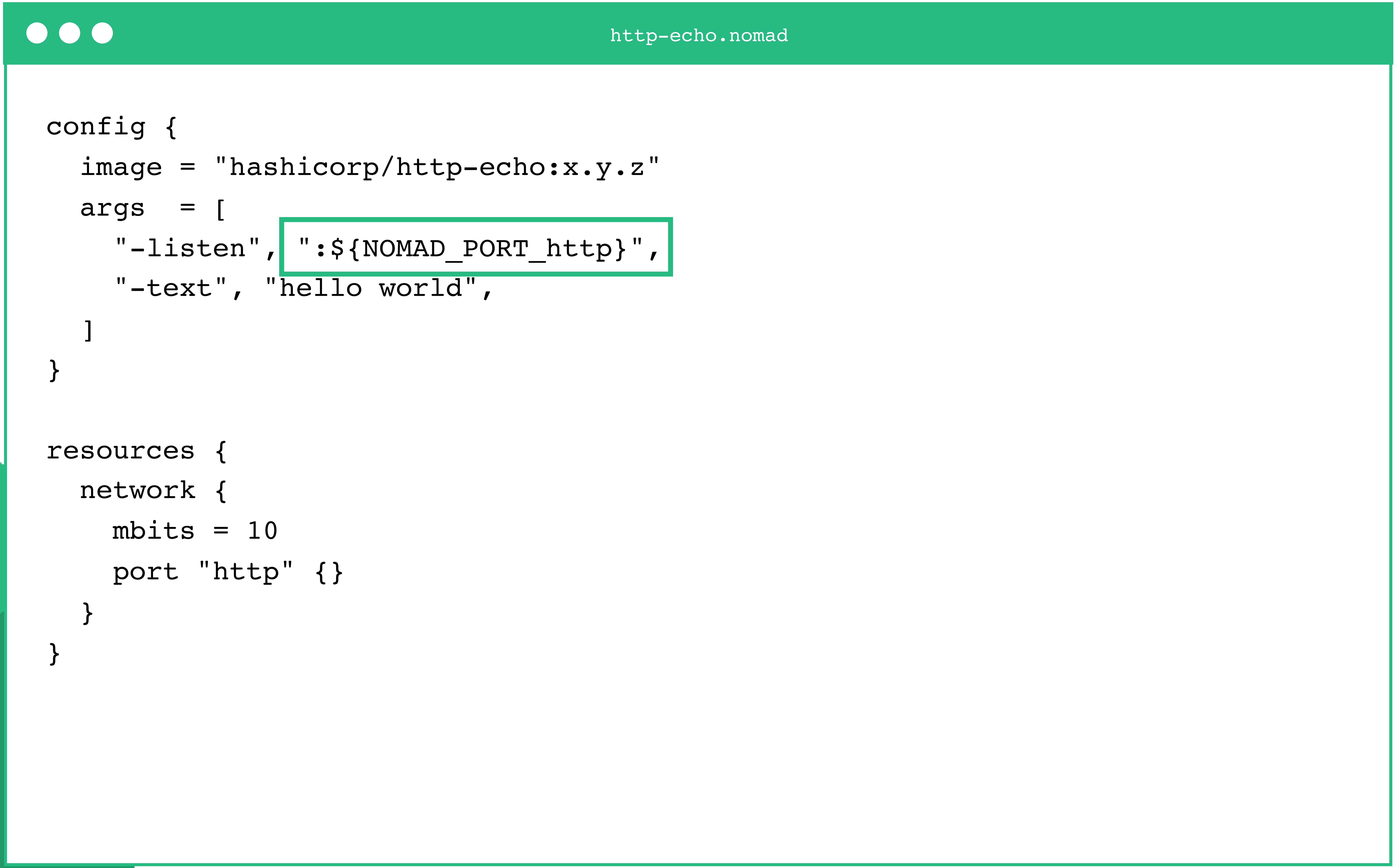
Nomad Interpolation

Nomad populates certain variables with information about the job.

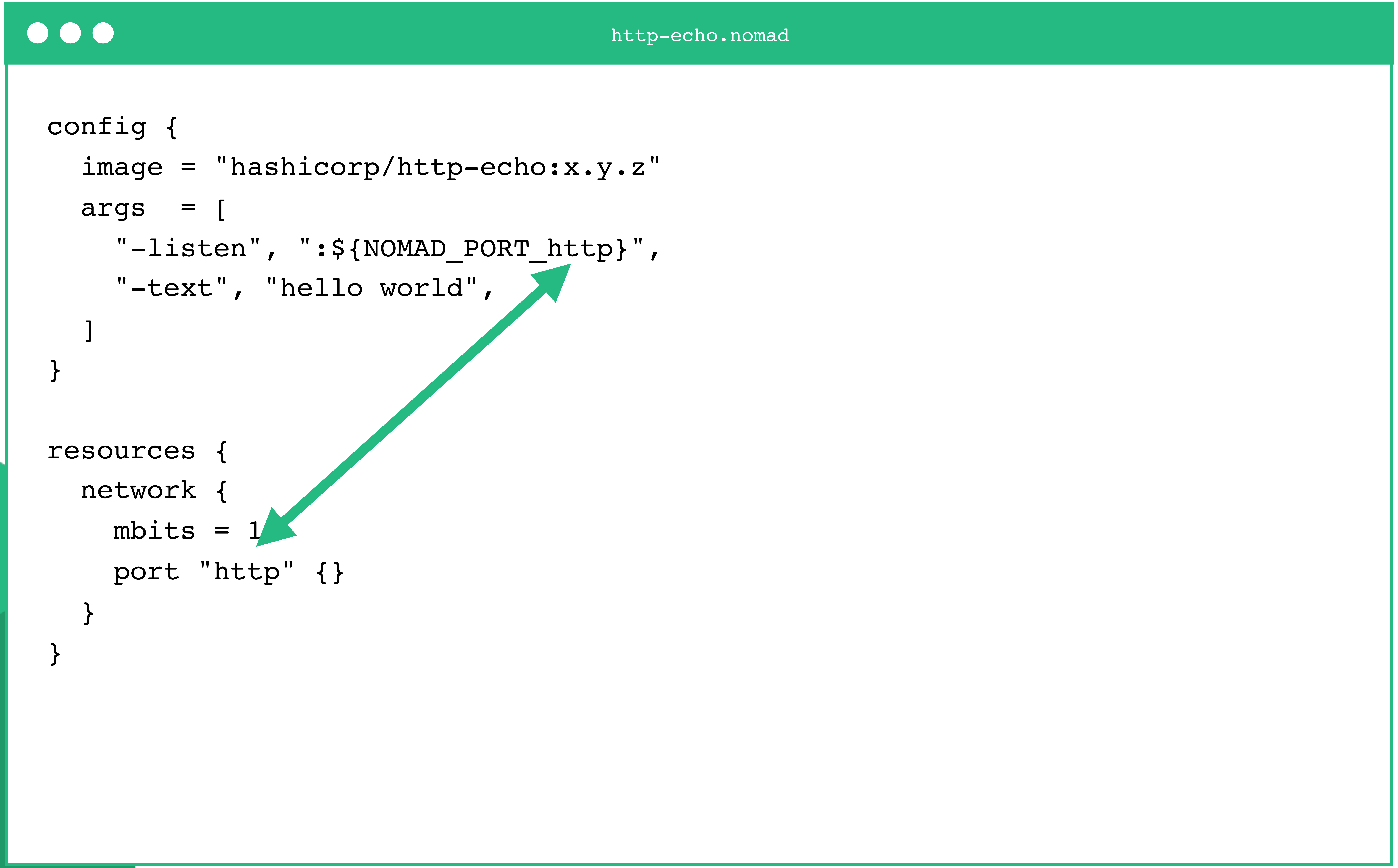
Values between `${}` are analyzed by the parser.



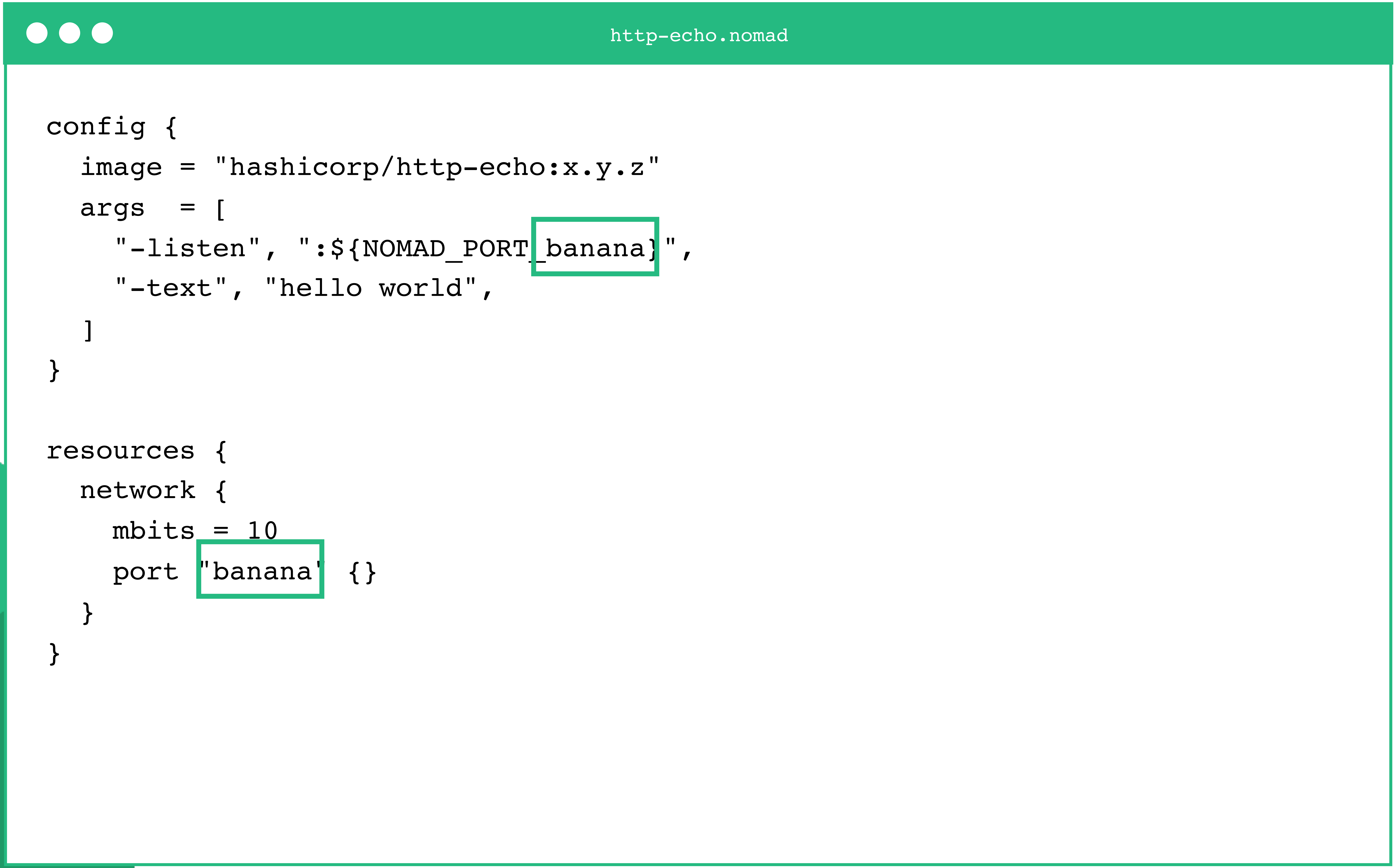
```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":??",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {}
  }
}
```



```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":${NOMAD_PORT_http}",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {}
  }
}
```



```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":${NOMAD_PORT_http}",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 1
    port "http" {}
  }
}
```



http-echo.nomad

```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":${NOMAD_PORT_banana}",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "banana" {}
  }
}
```

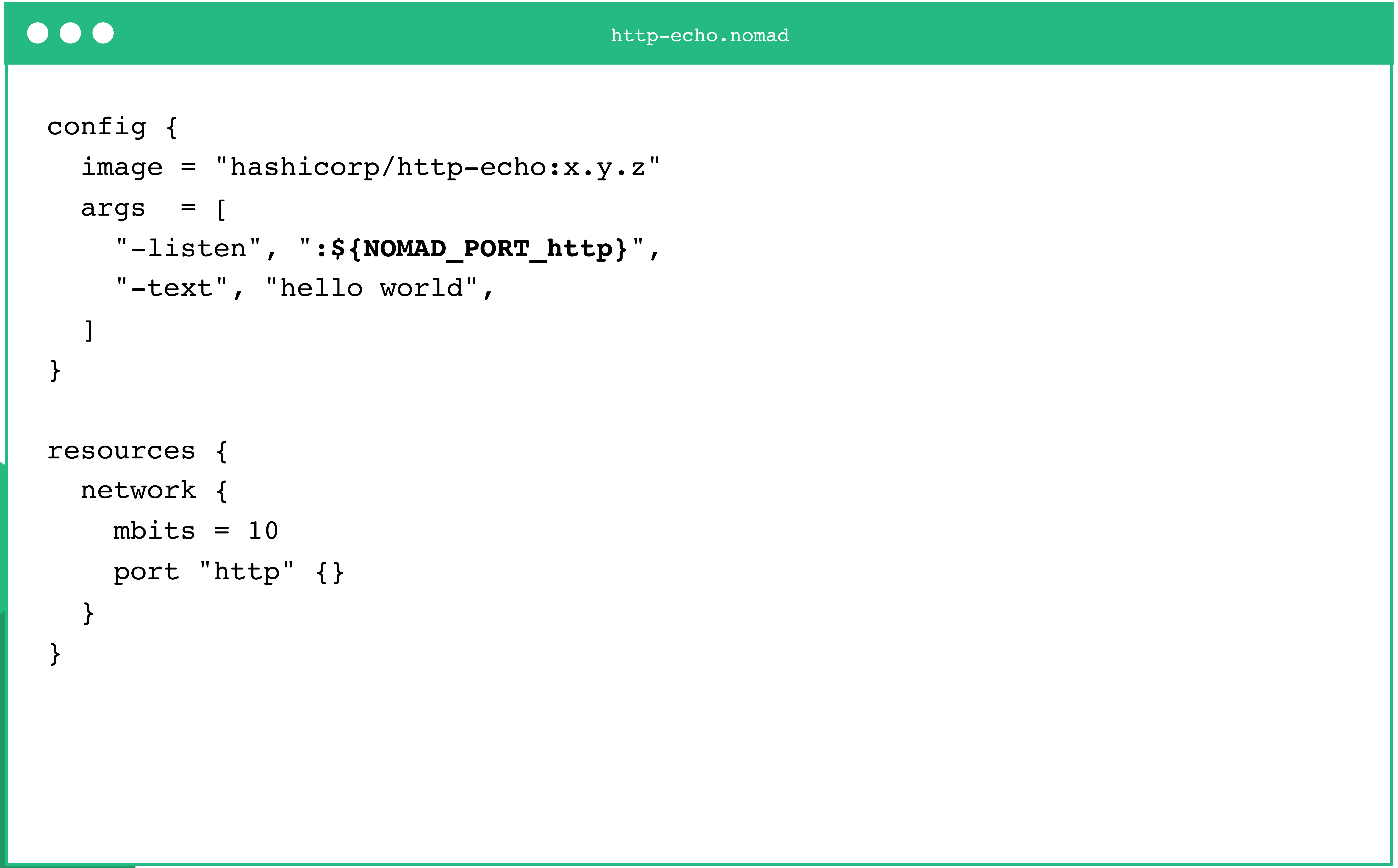
Exercise: Resubmit Job

Make these required changes to the job.

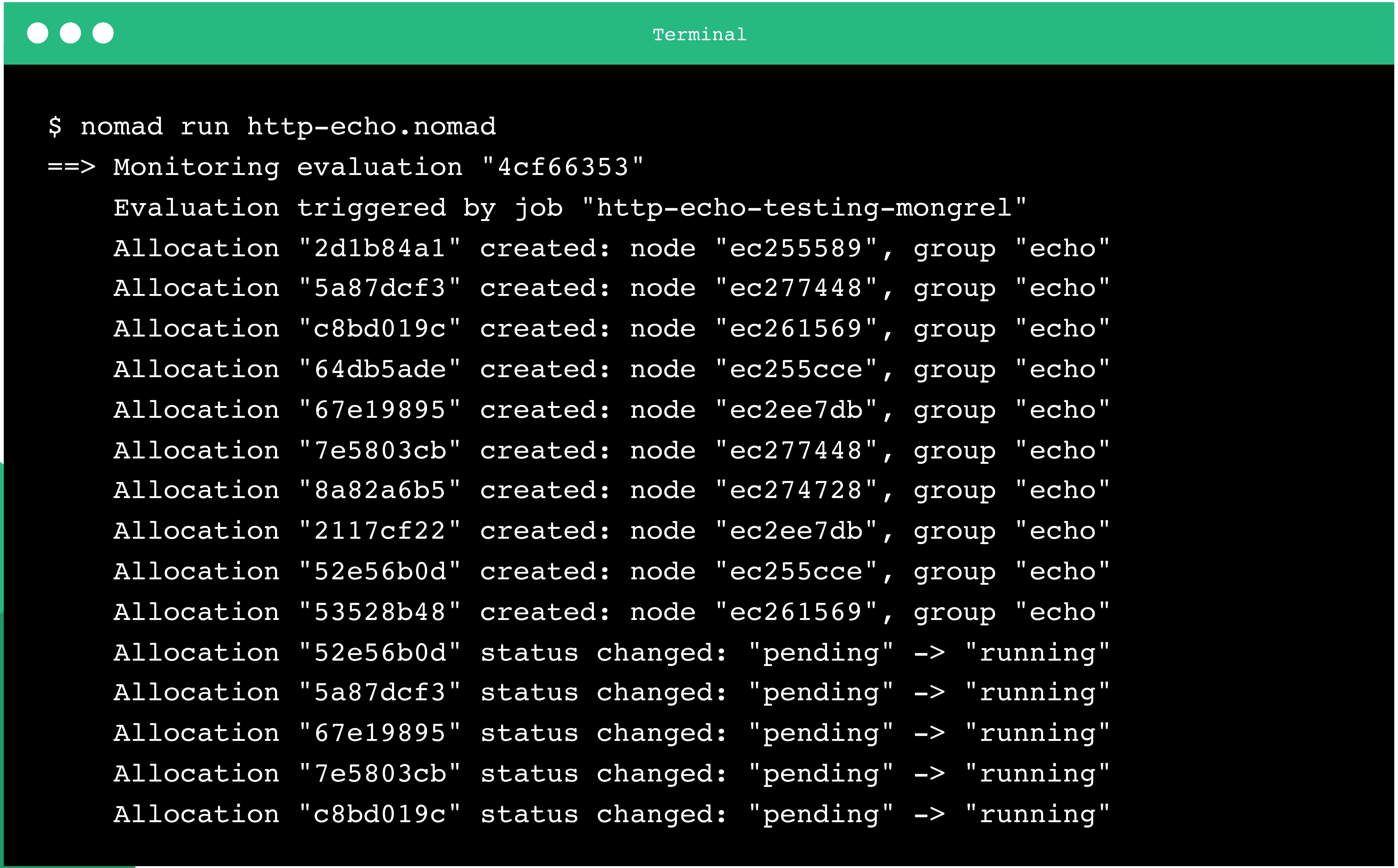
Resubmit the job to the Nomad server for evaluation.

Use `curl` to query the updated instance.

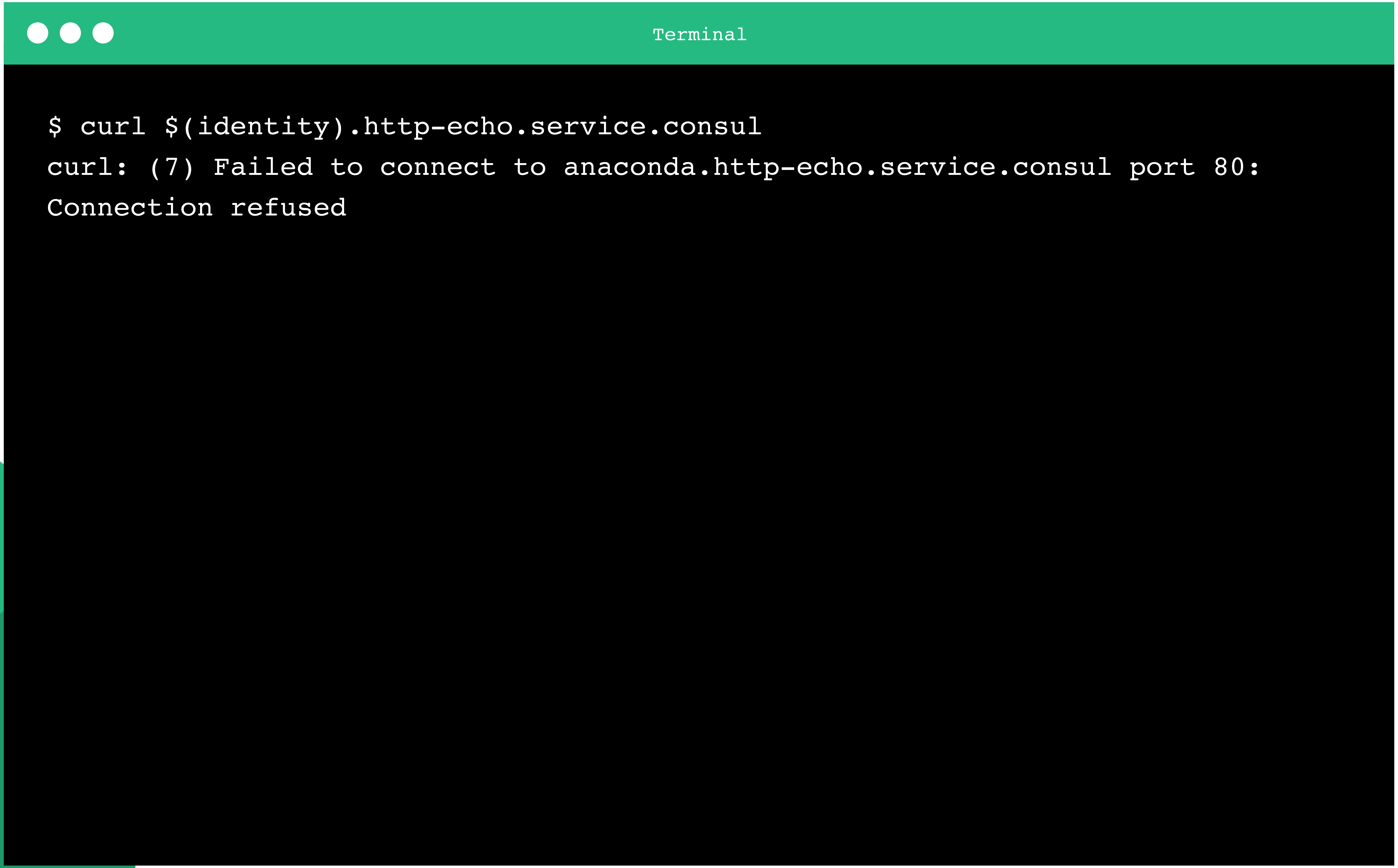
HINT: Google "nomad interpolation".



```
config {
  image = "hashicorp/http-echo:x.y.z"
  args  = [
    "-listen", ":${NOMAD_PORT_http}",
    "-text", "hello world",
  ]
}

resources {
  network {
    mbits = 10
    port "http" {}
  }
}
```



```
$ nomad run http-echo.nomad
==> Monitoring evaluation "4cf66353"
    Evaluation triggered by job "http-echo-testing-mongrel"
    Allocation "2d1b84a1" created: node "ec255589", group "echo"
    Allocation "5a87dcf3" created: node "ec277448", group "echo"
    Allocation "c8bd019c" created: node "ec261569", group "echo"
    Allocation "64db5ade" created: node "ec255cce", group "echo"
    Allocation "67e19895" created: node "ec2ee7db", group "echo"
    Allocation "7e5803cb" created: node "ec277448", group "echo"
    Allocation "8a82a6b5" created: node "ec274728", group "echo"
    Allocation "2117cf22" created: node "ec2ee7db", group "echo"
    Allocation "52e56b0d" created: node "ec255cce", group "echo"
    Allocation "53528b48" created: node "ec261569", group "echo"
    Allocation "52e56b0d" status changed: "pending" -> "running"
    Allocation "5a87dcf3" status changed: "pending" -> "running"
    Allocation "67e19895" status changed: "pending" -> "running"
    Allocation "7e5803cb" status changed: "pending" -> "running"
    Allocation "c8bd019c" status changed: "pending" -> "running"
```



```
$ curl $(identity).http-echo.service.consul
curl: (7) Failed to connect to anaconda.http-echo.service.consul port 80:
Connection refused
```

Why is the Job Inaccessible?

Previously our job was hard-bound to port 80 (default HTTP port).

Now Nomad is dynamically allocating a high-numbered port, so our service could be listening on any port.

Must rely on service discovery to find the port (which we will discuss later).

Monitoring Jobs

Command: `nomad status`

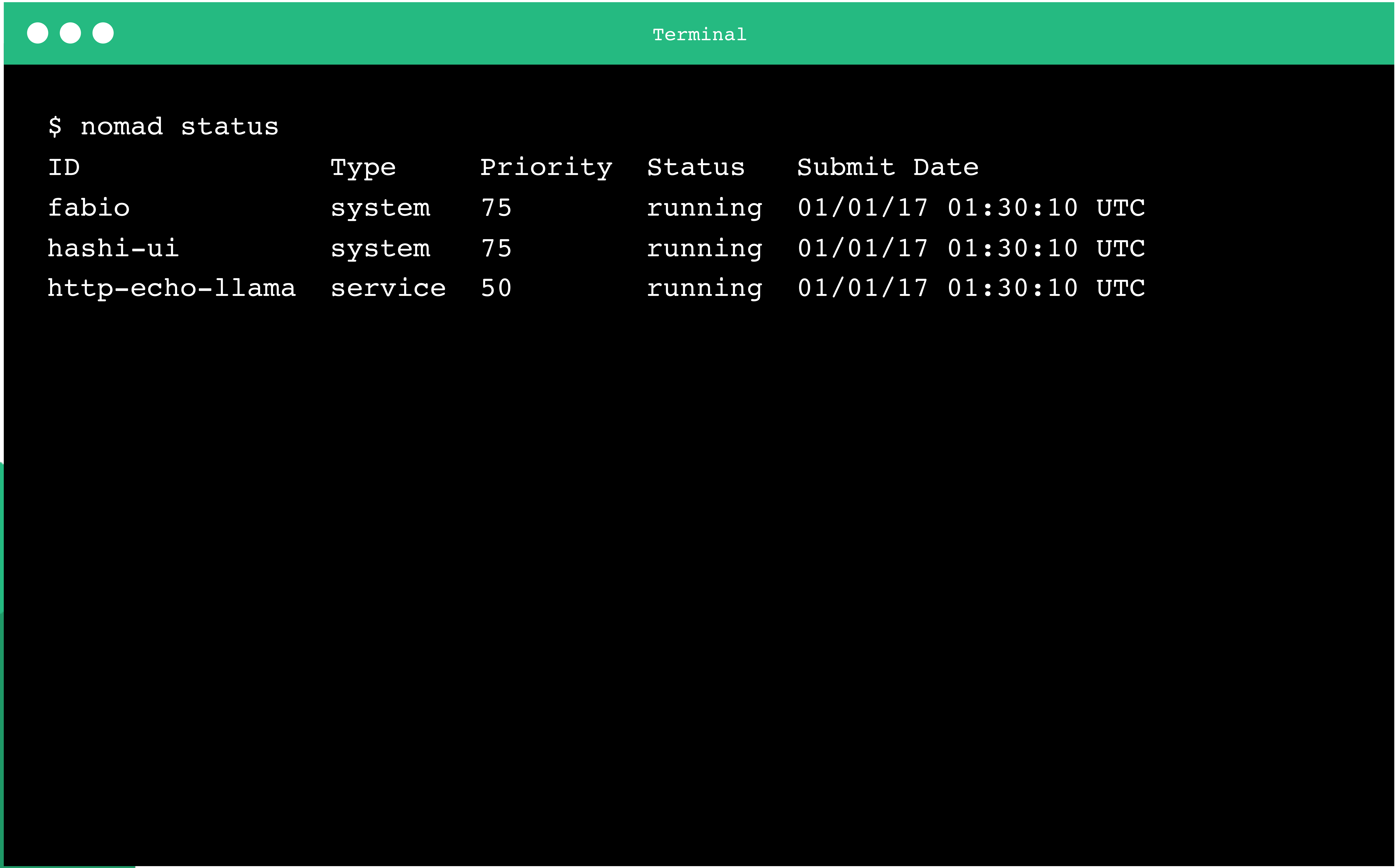
The **`nomad status`** command lists the status of all jobs in the system.

If supplied an optional argument, the **`nomad status`** command lists detailed information about the job name.

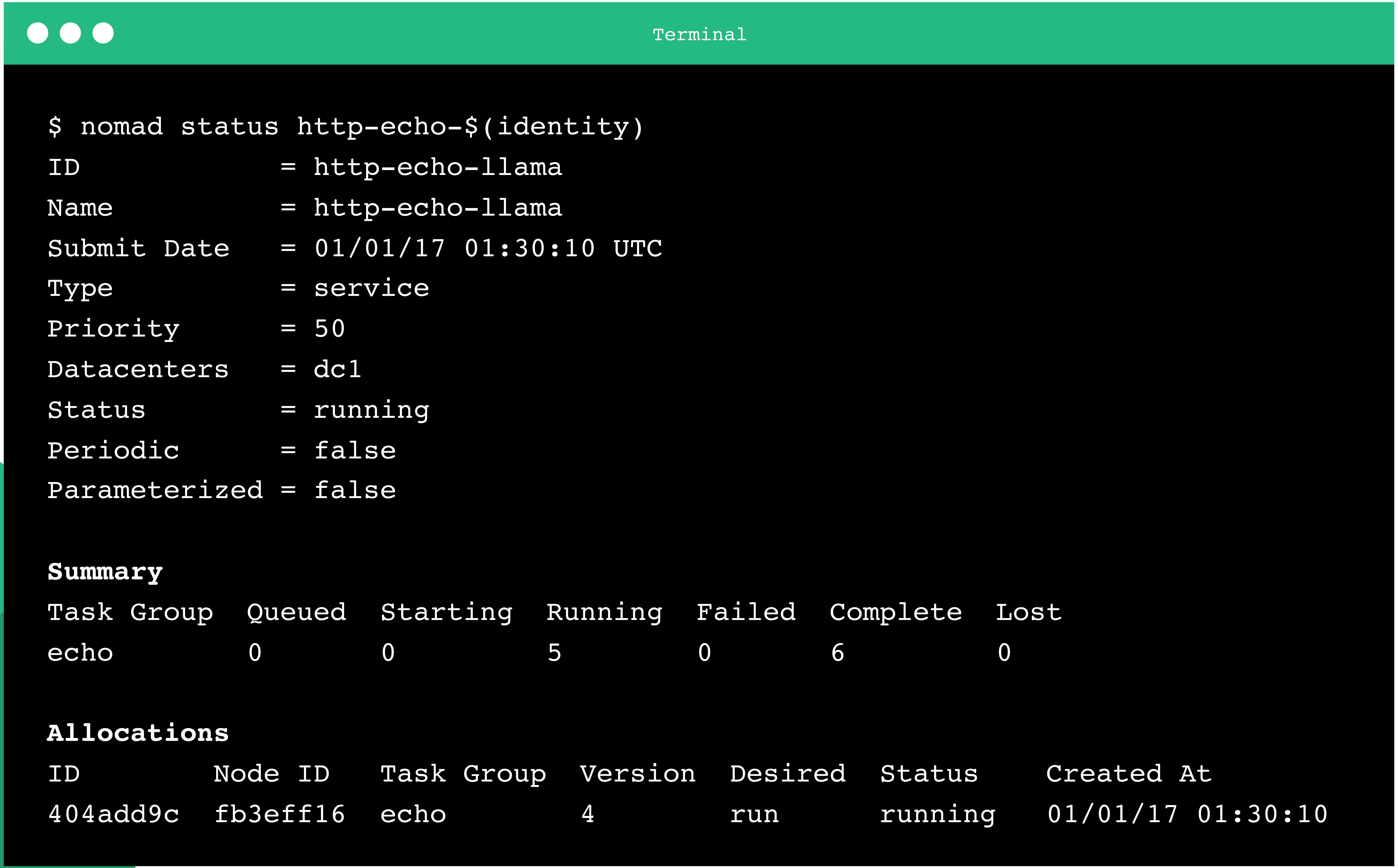
Exercise: Run `nomad status`

Run the **`nomad status`** command to see the status of all jobs in the system.

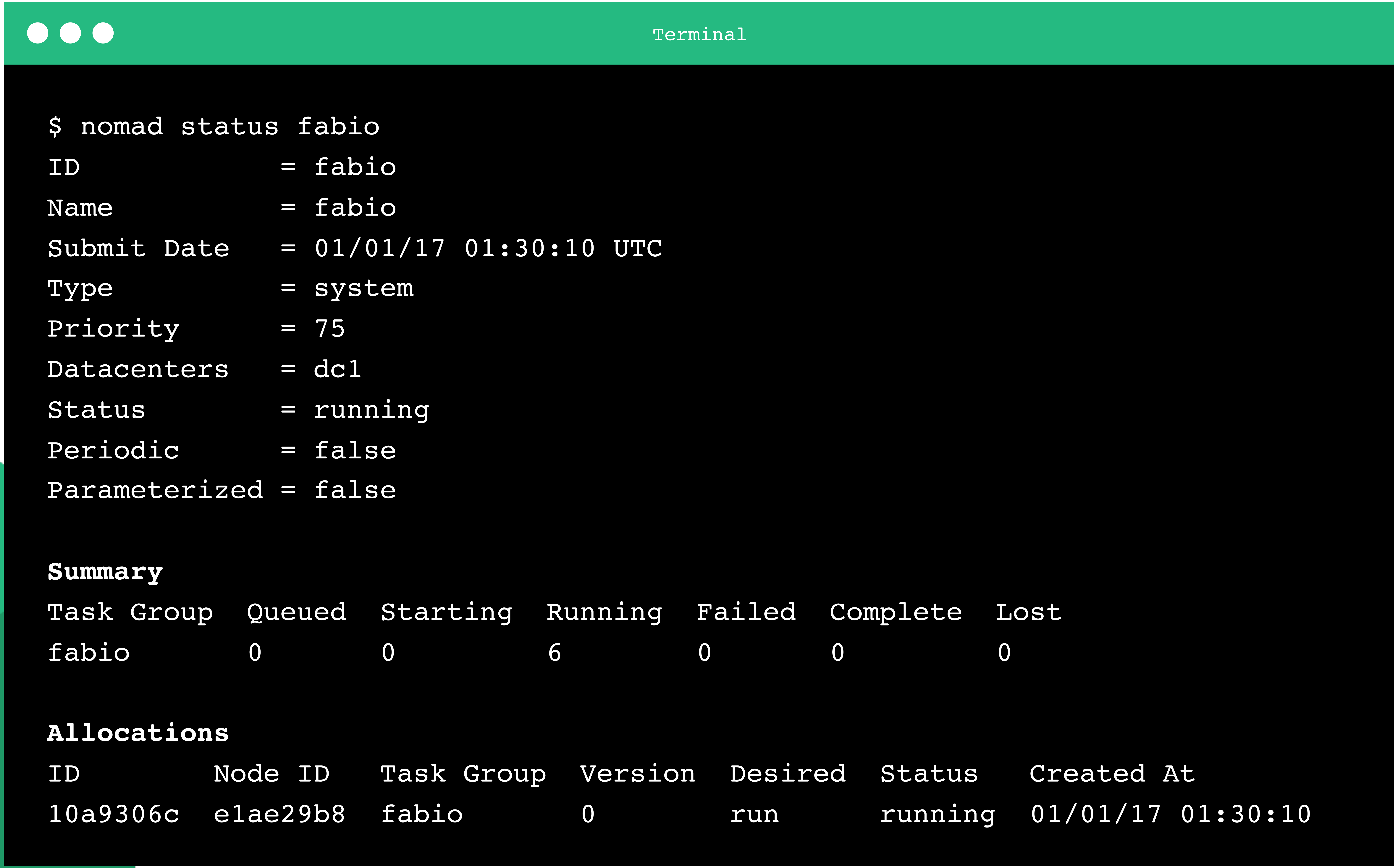
Run the **`nomad status`** command with the name of *your* job to get detailed job information.



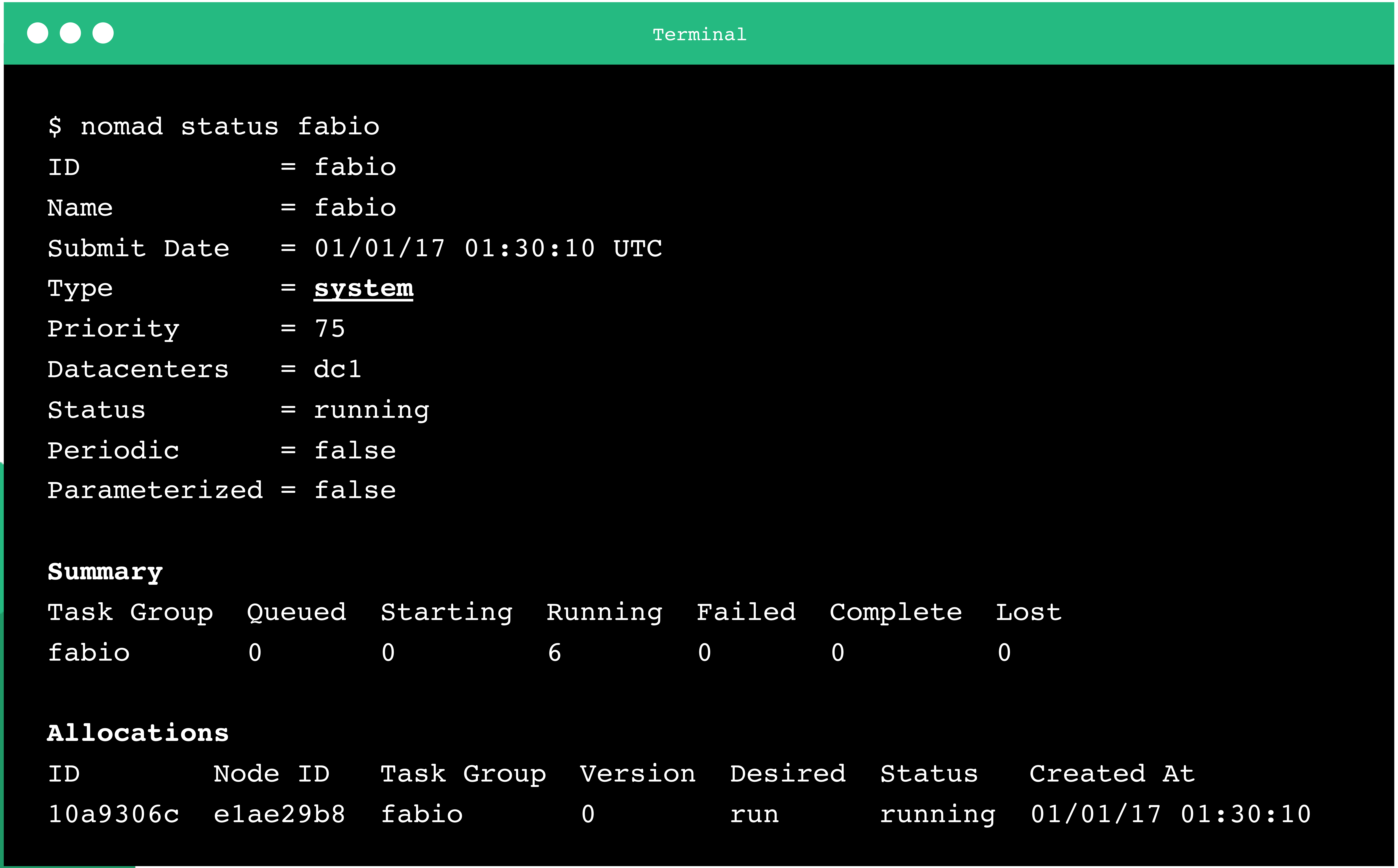
```
$ nomad status
ID                 Type     Priority  Status   Submit Date
fabio              system   75        running  01/01/17 01:30:10 UTC
hashi-ui           system   75        running  01/01/17 01:30:10 UTC
http-echo-llama    service  50        running  01/01/17 01:30:10 UTC
```



```
$ nomad status http-echo-$(identity)
ID            = http-echo-llama
Name          = http-echo-llama
Submit Date   = 01/01/17 01:30:10 UTC
Type          = service
Priority      = 50
Datacenters   = dc1
Status        = running
Periodic      = false
Parameterized = false

Summary
Task Group  Queued  Starting  Running  Failed  Complete  Lost
echo        0       0         5        0       6         0

Allocations
ID        Node ID   Task Group  Version  Desired  Status   Created At
404add9c  fb3eff16  echo        4        run      running  01/01/17 01:30:10
```



```
$ nomad status fabio
ID            = fabio
Name          = fabio
Submit Date   = 01/01/17 01:30:10 UTC
Type          = system
Priority      = 75
Datacenters   = dc1
Status        = running
Periodic      = false
Parameterized = false

Summary
Task Group  Queued  Starting  Running  Failed  Complete  Lost
fabio       0       0         6        0       0         0

Allocations
ID        Node ID   Task Group  Version  Desired  Status   Created At
10a9306c  e1ae29b8  fabio       0        run      running  01/01/17 01:30:10
```



```
$ nomad status fabio
ID            = fabio
Name          = fabio
Submit Date   = 01/01/17 01:30:10 UTC
Type          = system
Priority      = 75
Datacenters   = dc1
Status        = running
Periodic      = false
Parameterized = false

Summary
Task Group  Queued  Starting  Running  Failed  Complete  Lost
fabio       0       0         6        0       0         0

Allocations
ID        Node ID   Task Group  Version  Desired  Status   Created At
10a9306c  e1ae29b8  fabio       0        run      running  01/01/17 01:30:10
```

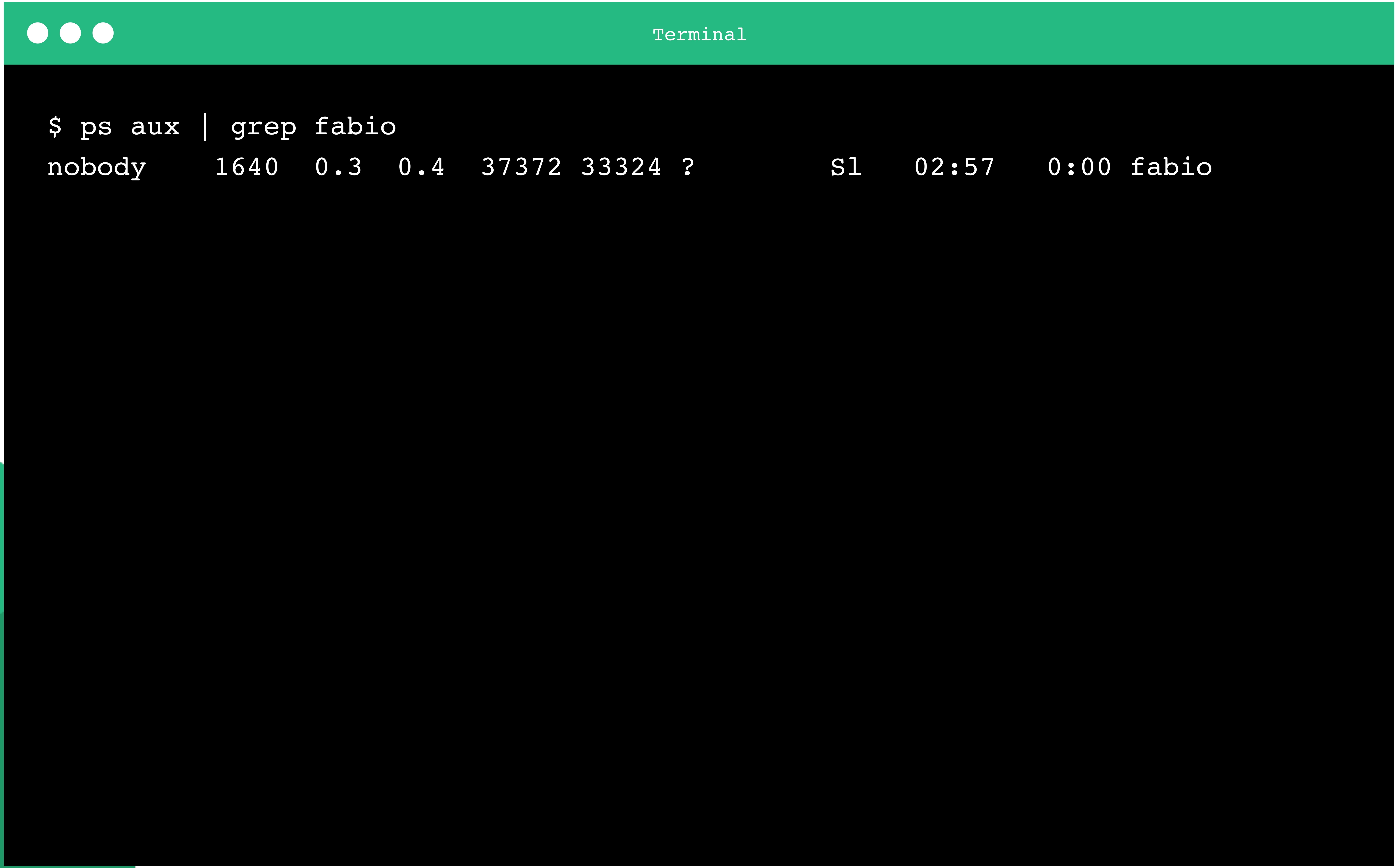
HashiCorp

Exercise: Verify Fabio is Running

We see fabio is running under Nomad.

Fabio is running as a local-exec job (outside of Docker).

Manually verify fabio is running by running `ps`.



```
$ ps aux | grep fabio
nobody     1640  0.3  0.4  37372 33324 ?        Sl   02:57   0:00 fabio
```

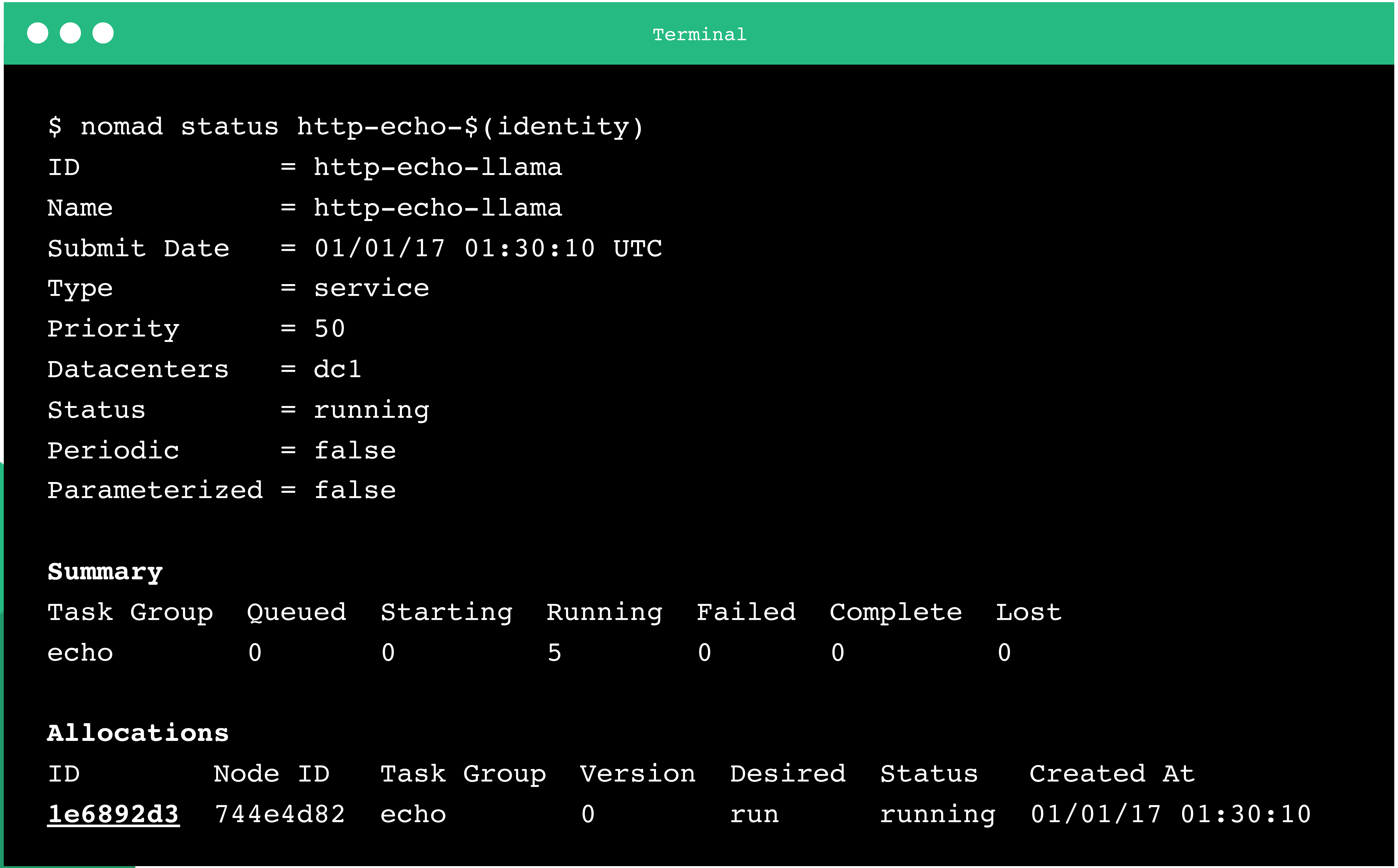
Command: `nomad alloc-status`

The `nomad alloc status` displays information about the given allocation ID, including run status, metadata, and failure messages.

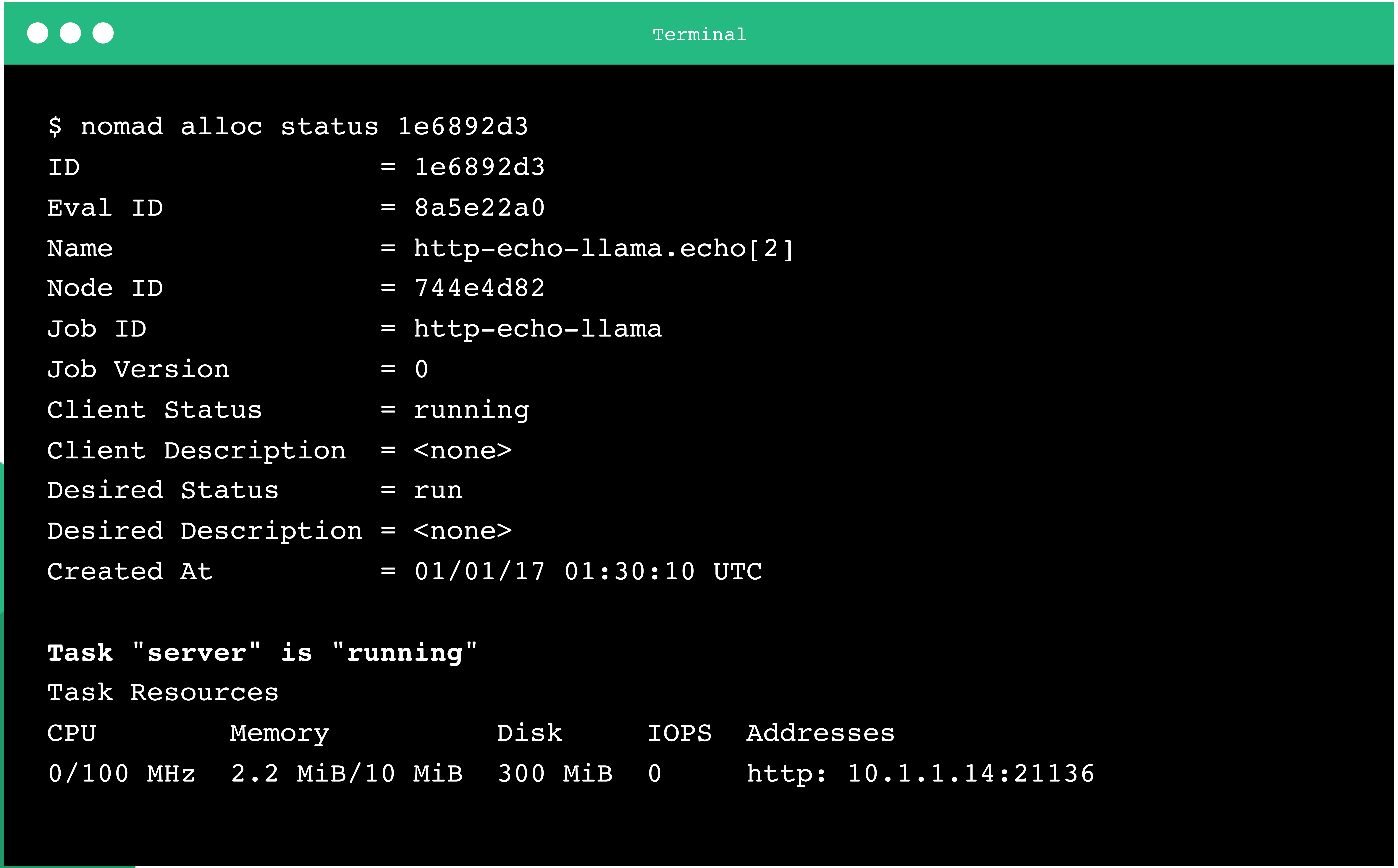
Exercise: Query Nomad Allocation Status

Find the ID of a running allocation using the `nomad status` command.

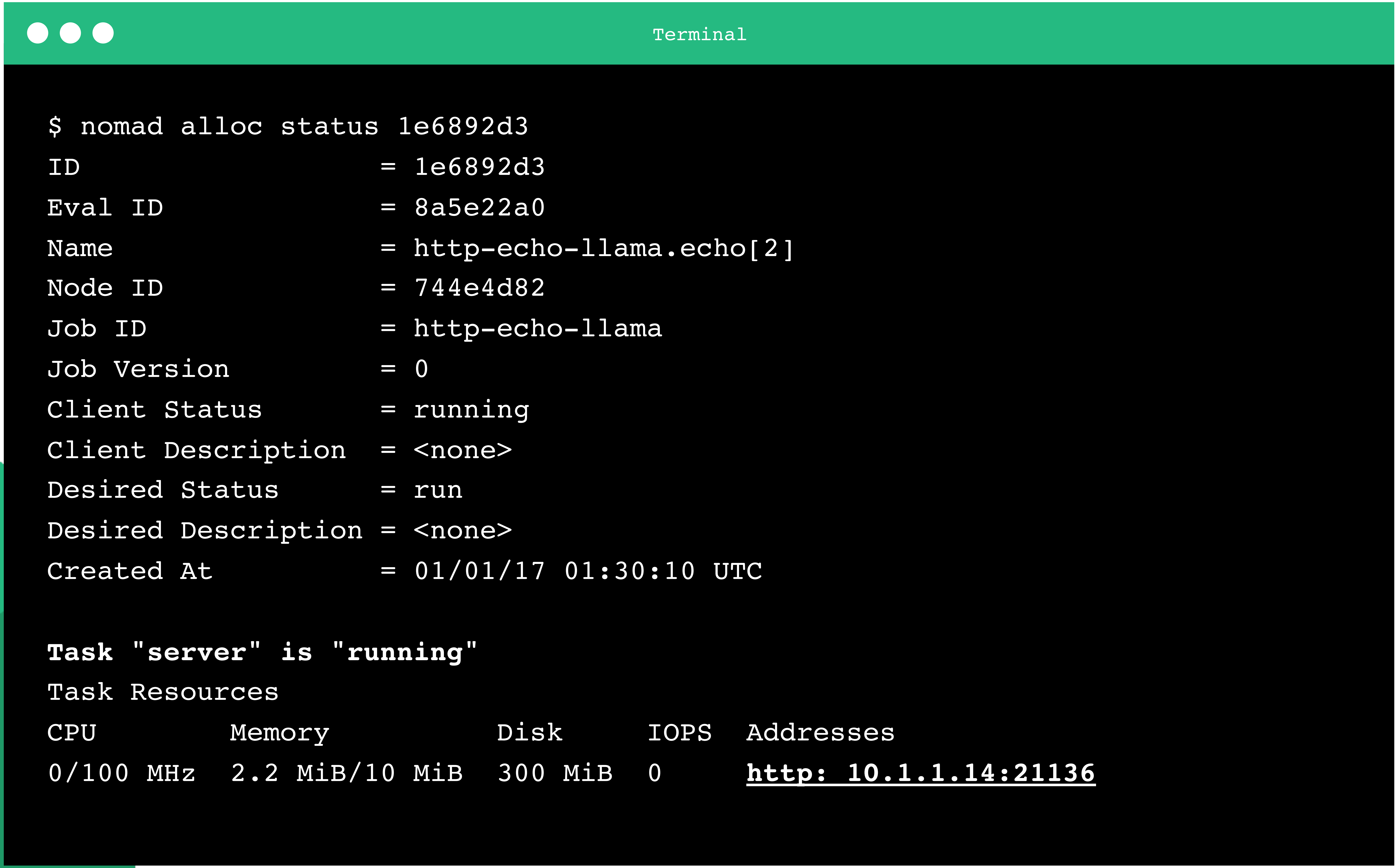
Query that allocation status using the `nomad alloc status` command with the allocation ID.



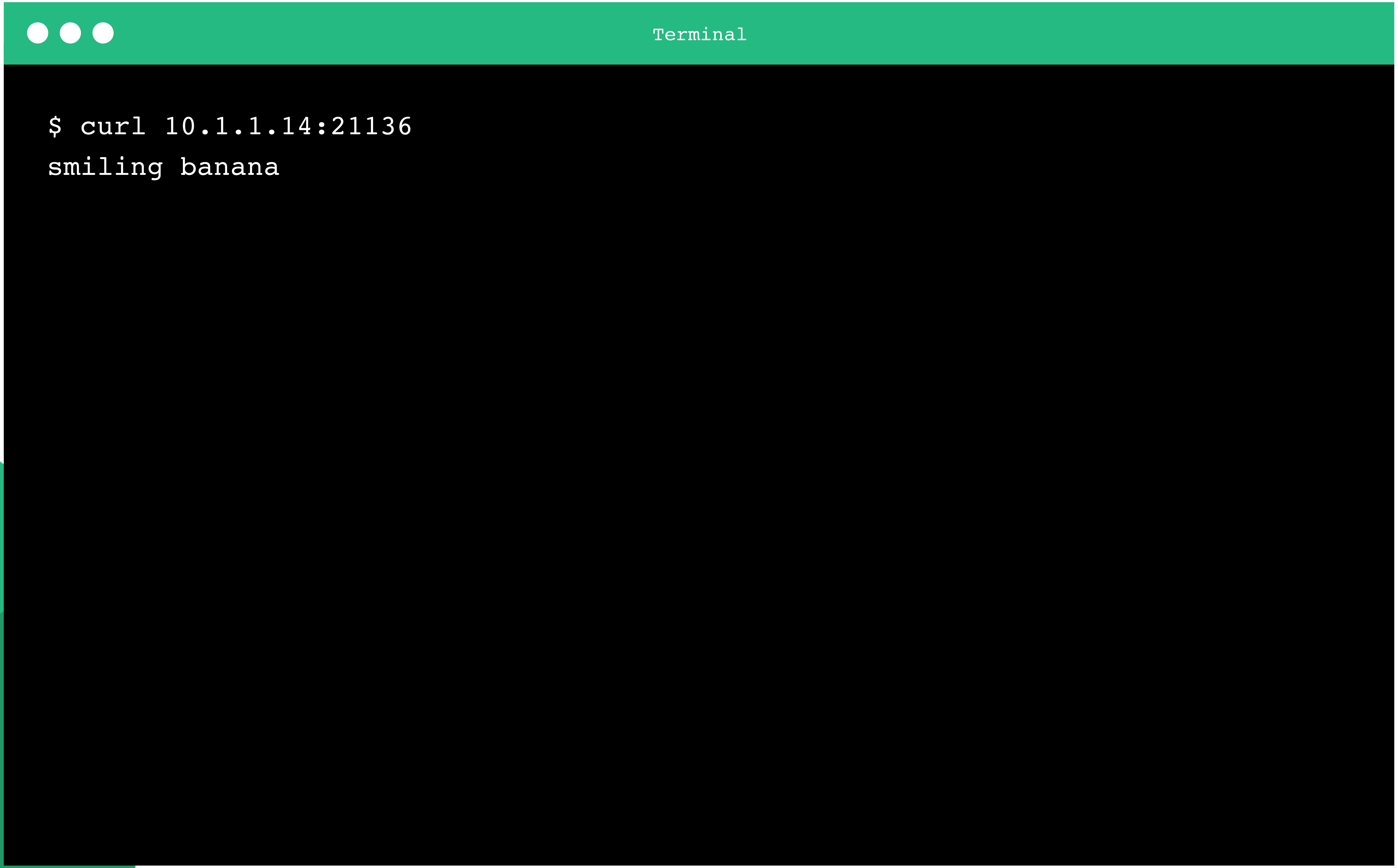
```
$ nomad status http-echo-$(identity)
ID            = http-echo-llama
Name          = http-echo-llama
Submit Date   = 01/01/17 01:30:10 UTC
Type          = service
Priority      = 50
Datacenters   = dc1
Status        = running
Periodic      = false
Parameterized = false

Summary
Task Group  Queued  Starting  Running  Failed  Complete  Lost
echo        0       0         5        0       0         0

Allocations
ID        Node ID   Task Group  Version  Desired  Status   Created At
1e6892d3  744e4d82  echo        0        run      running  01/01/17 01:30:10
```



```
$ nomad alloc status 1e6892d3
ID                  = 1e6892d3
Eval ID             = 8a5e22a0
Name                = http-echo-llama.echo[2]
Node ID             = 744e4d82
Job ID              = http-echo-llama
Job Version         = 0
Client Status       = running
Client Description  = <none>
Desired Status      = run
Desired Description = <none>
Created At          = 01/01/17 01:30:10 UTC

Task "server" is "running"
Task Resources
CPU        Memory          Disk     IOPS  Addresses
0/100 MHz  2.2 MiB/10 MiB  300 MiB  0     http: 10.1.1.14:21136
```



```
$ nomad alloc status 1e6892d3
ID                  = 1e6892d3
Eval ID             = 8a5e22a0
Name                = http-echo-llama.echo[2]
Node ID             = 744e4d82
Job ID              = http-echo-llama
Job Version         = 0
Client Status       = running
Client Description  = <none>
Desired Status      = run
Desired Description = <none>
Created At          = 01/01/17 01:30:10 UTC

Task "server" is "running"
Task Resources
CPU         Memory           Disk     IOPS  Addresses
0/100 MHz   2.2 MiB/10 MiB   300 MiB  0     http: 10.1.1.14:21136
```

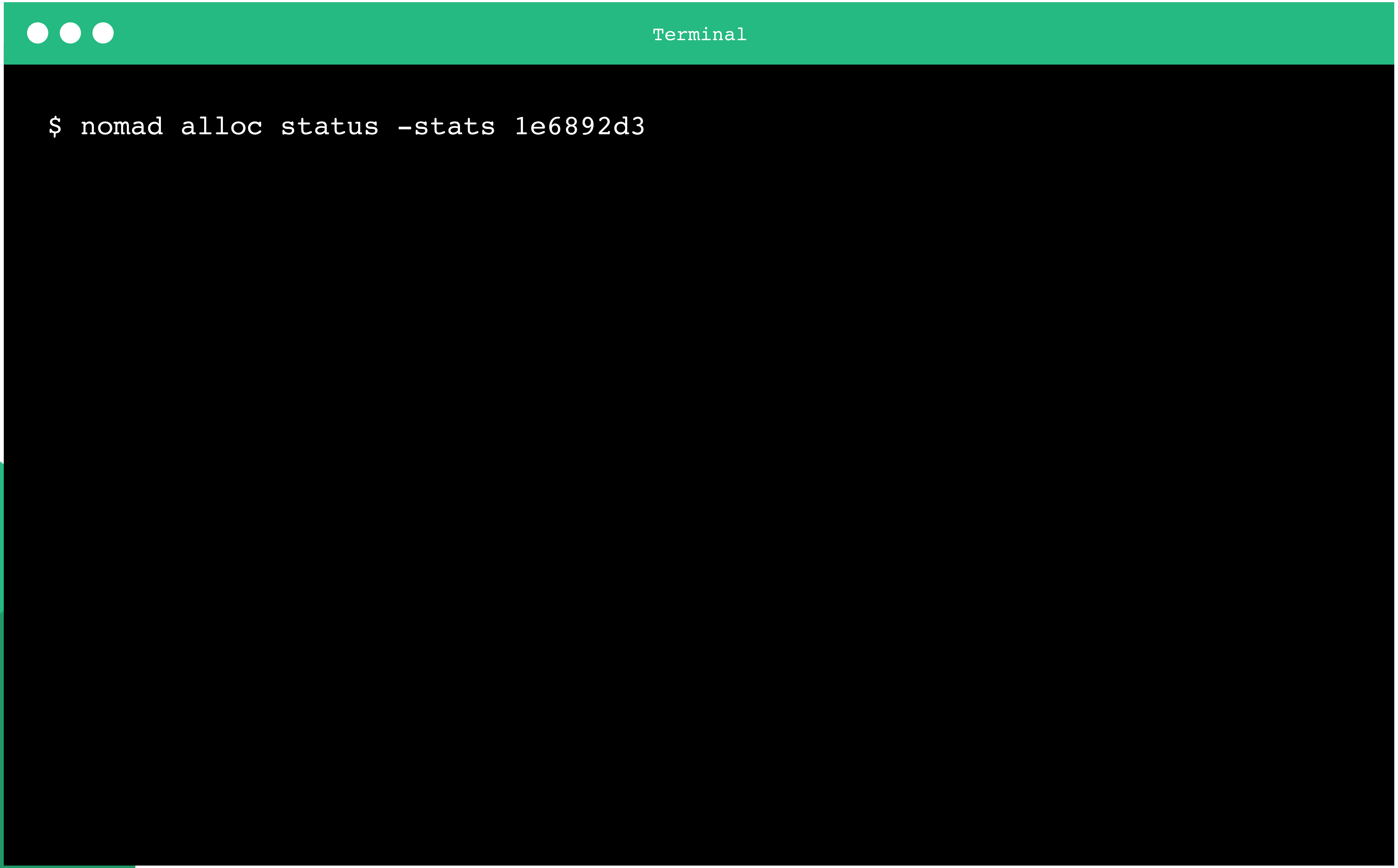


```
$ curl 10.1.1.14:21136
smiling banana
```

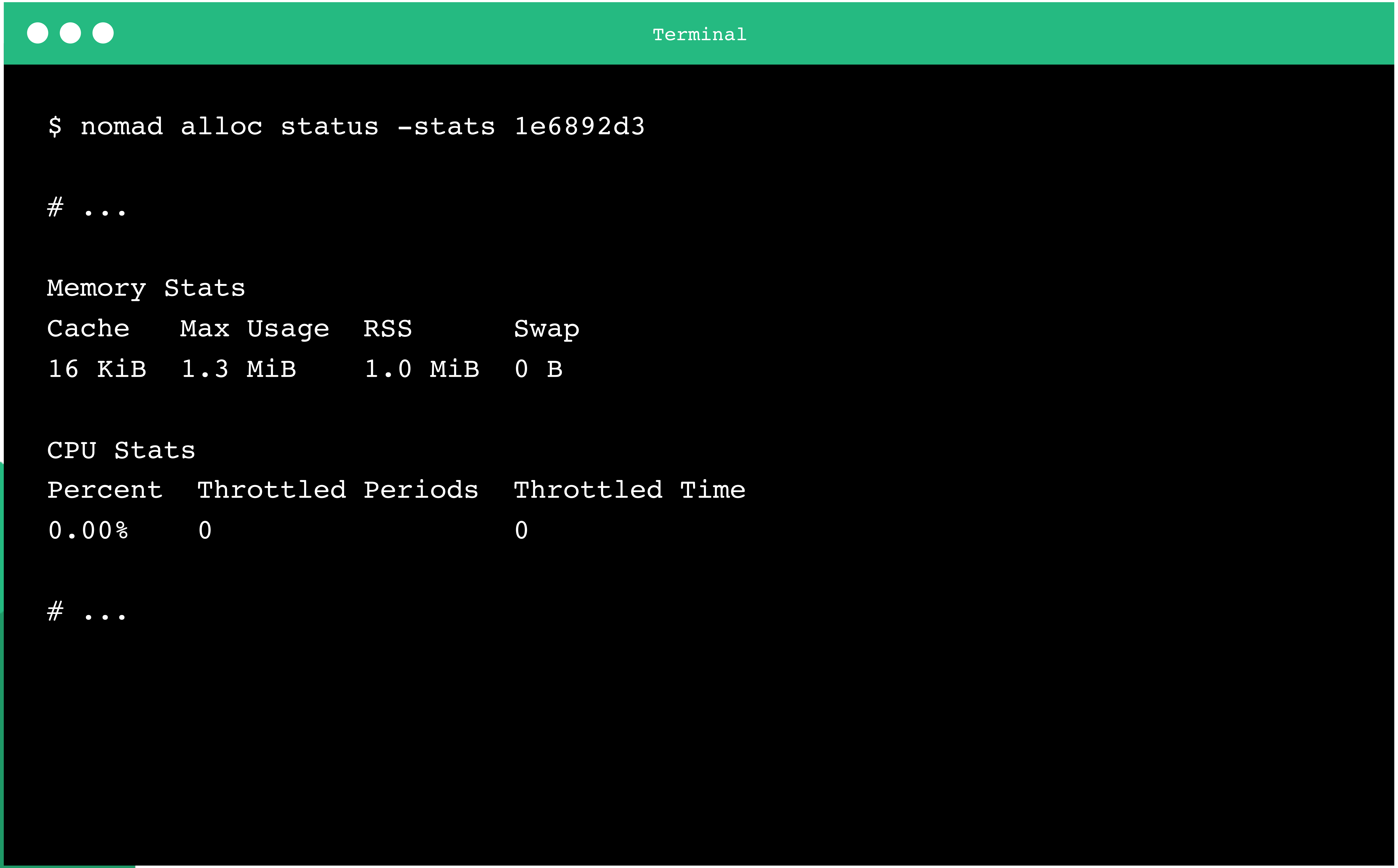
Exercise: Query Nomad Allocation Stats

Find and display the detailed resource statistics on your allocation.

HINT: You may need to pass an extra flag to the `alloc status` command.



```
$ nomad alloc status -stats 1e6892d3
```



```
$ nomad alloc status -stats 1e6892d3

# ...

Memory Stats
Cache    Max Usage  RSS       Swap
16 KiB   1.3 MiB    1.0 MiB   0 B

CPU Stats
Percent  Throttled Periods  Throttled Time
0.00%    0                  0

# ...
```

Command: `nomad logs`

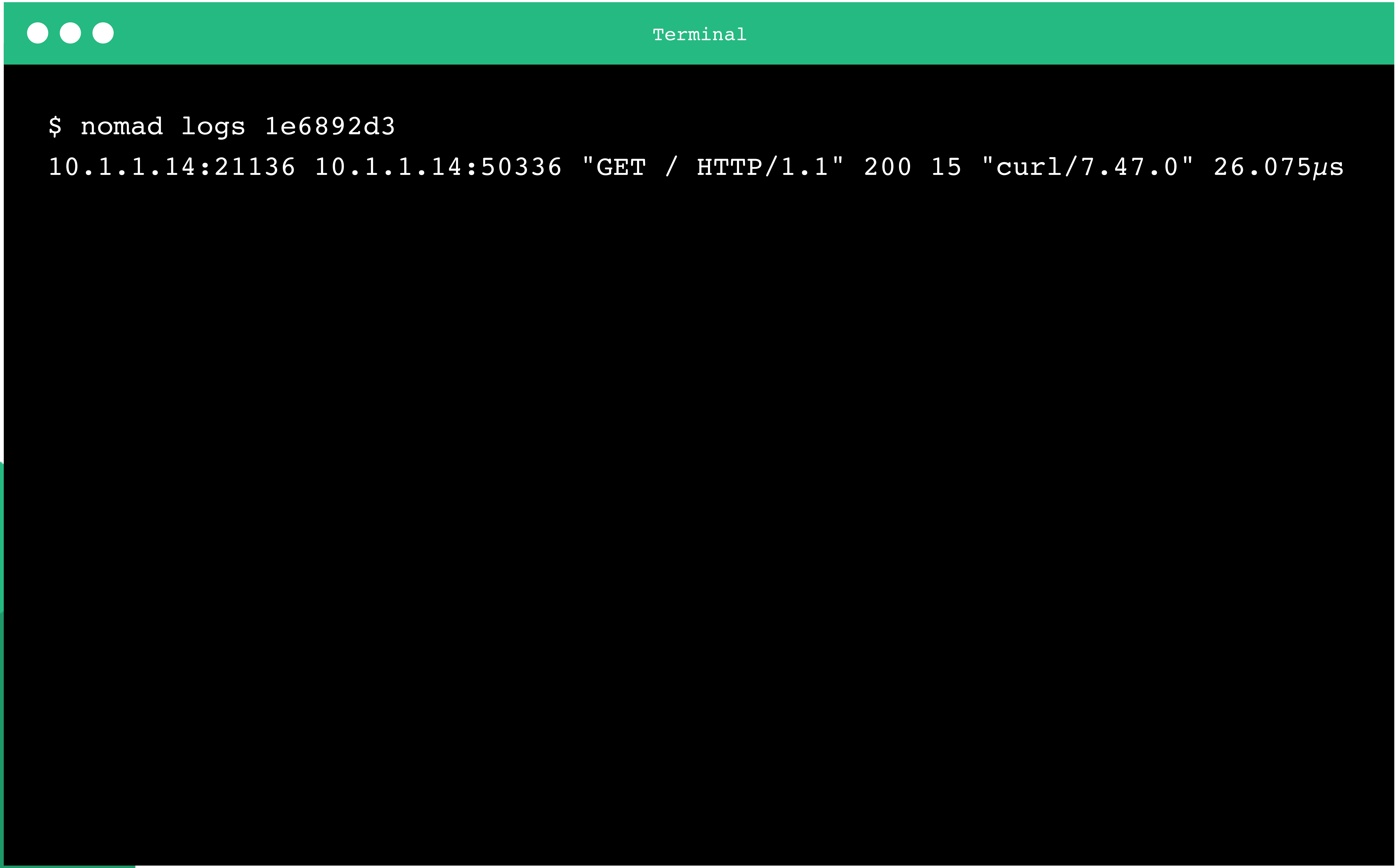
The `nomad logs` command can query the stdout and stderr from your task

Requires an allocation ID

Exercise: Run `nomad logs`

Run `nomad status` to get an allocation ID of a running job (or use the previous one).

Run the `nomad logs` command to view the most recent output for that allocation.



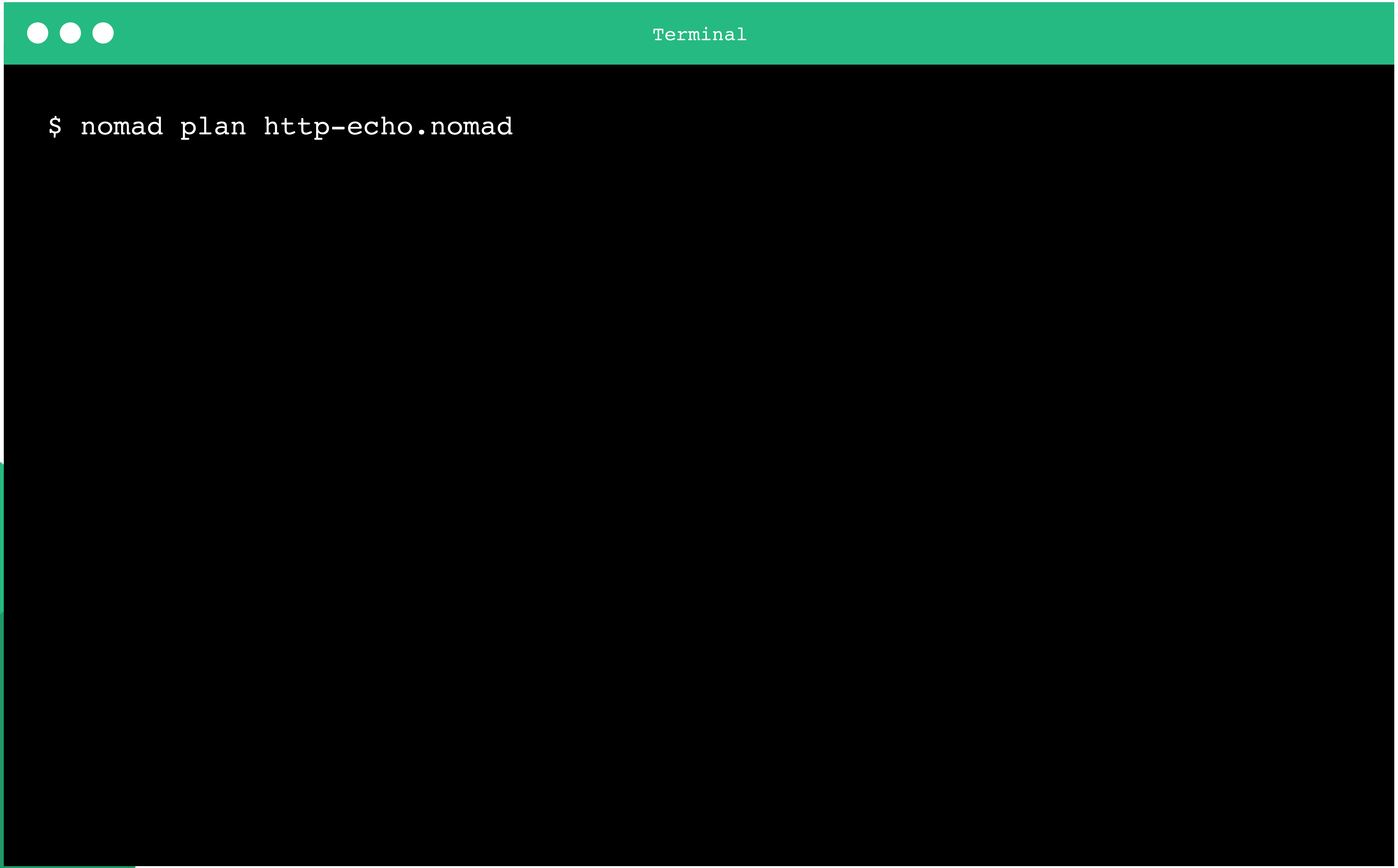
```
$ nomad logs 1e6892d3
10.1.1.14:21136 10.1.1.14:50336 "GET / HTTP/1.1" 200 15 "curl/7.47.0" 26.075µs
```

Resources, Constraints, and Planning

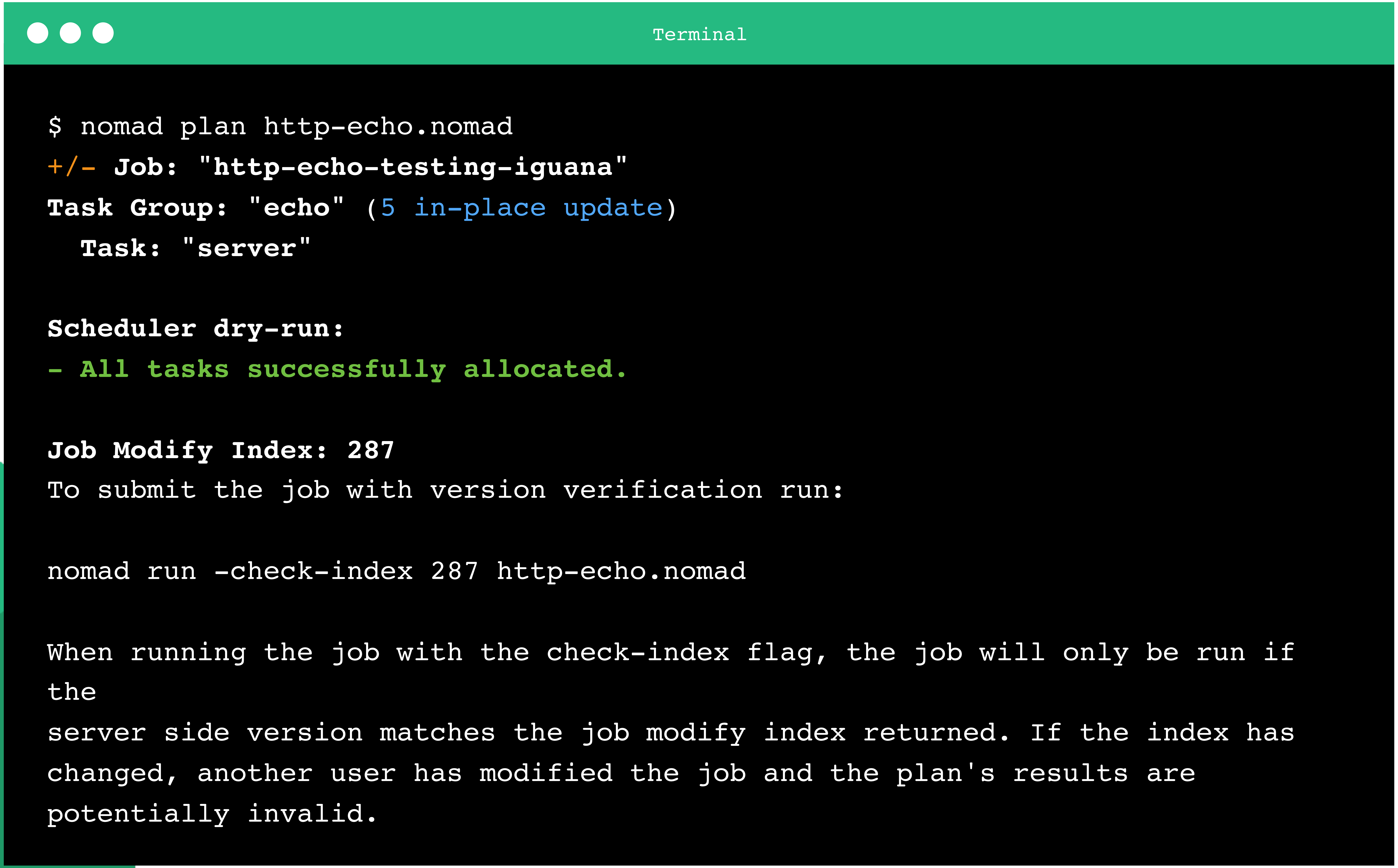
Command: `nomad plan`

The **`nomad plan`** command invokes the scheduler in a dry-run mode to show you what will happen if the job was submitted.

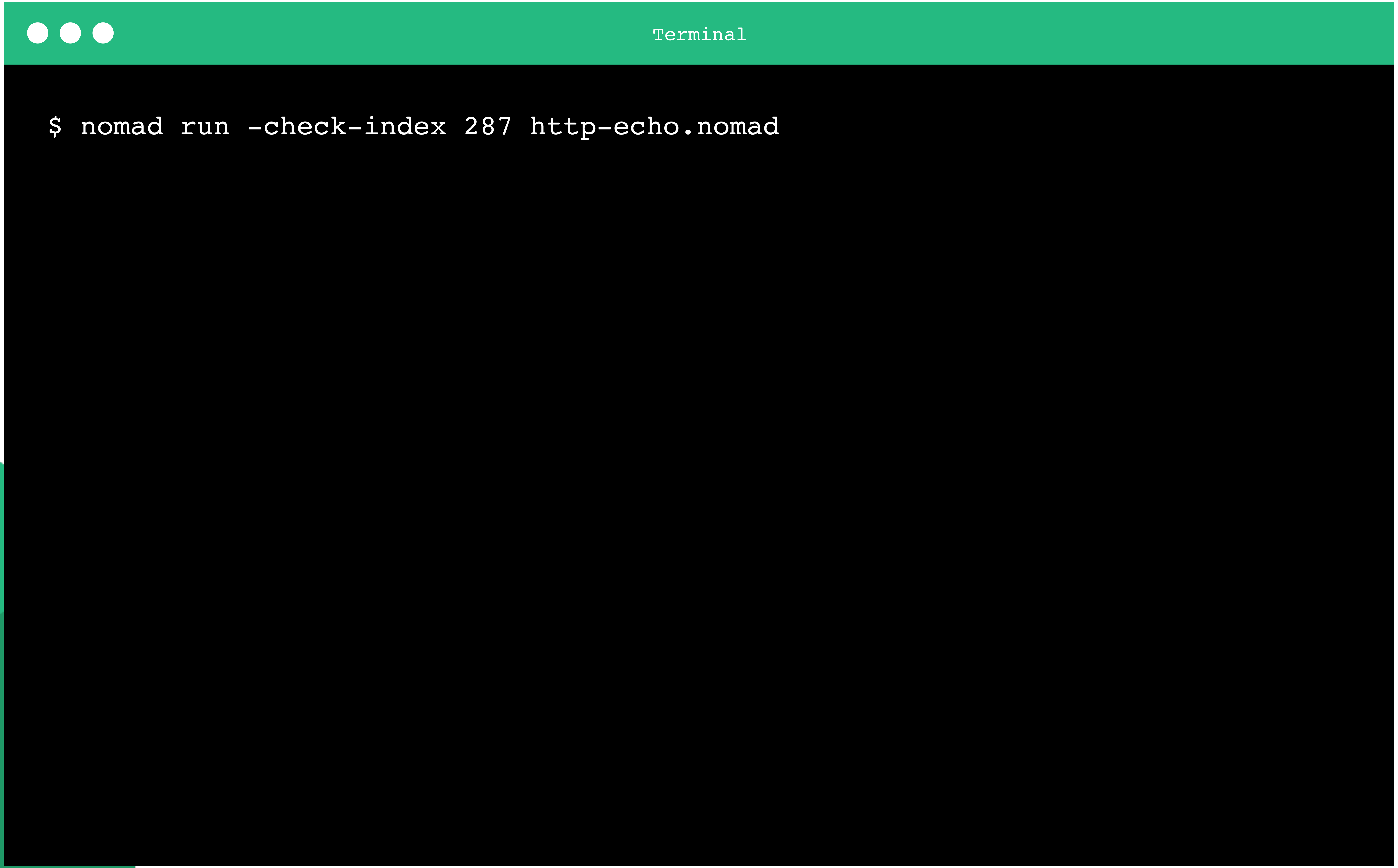
The resulting index can be specified when running the job to ensure no changes have happened.



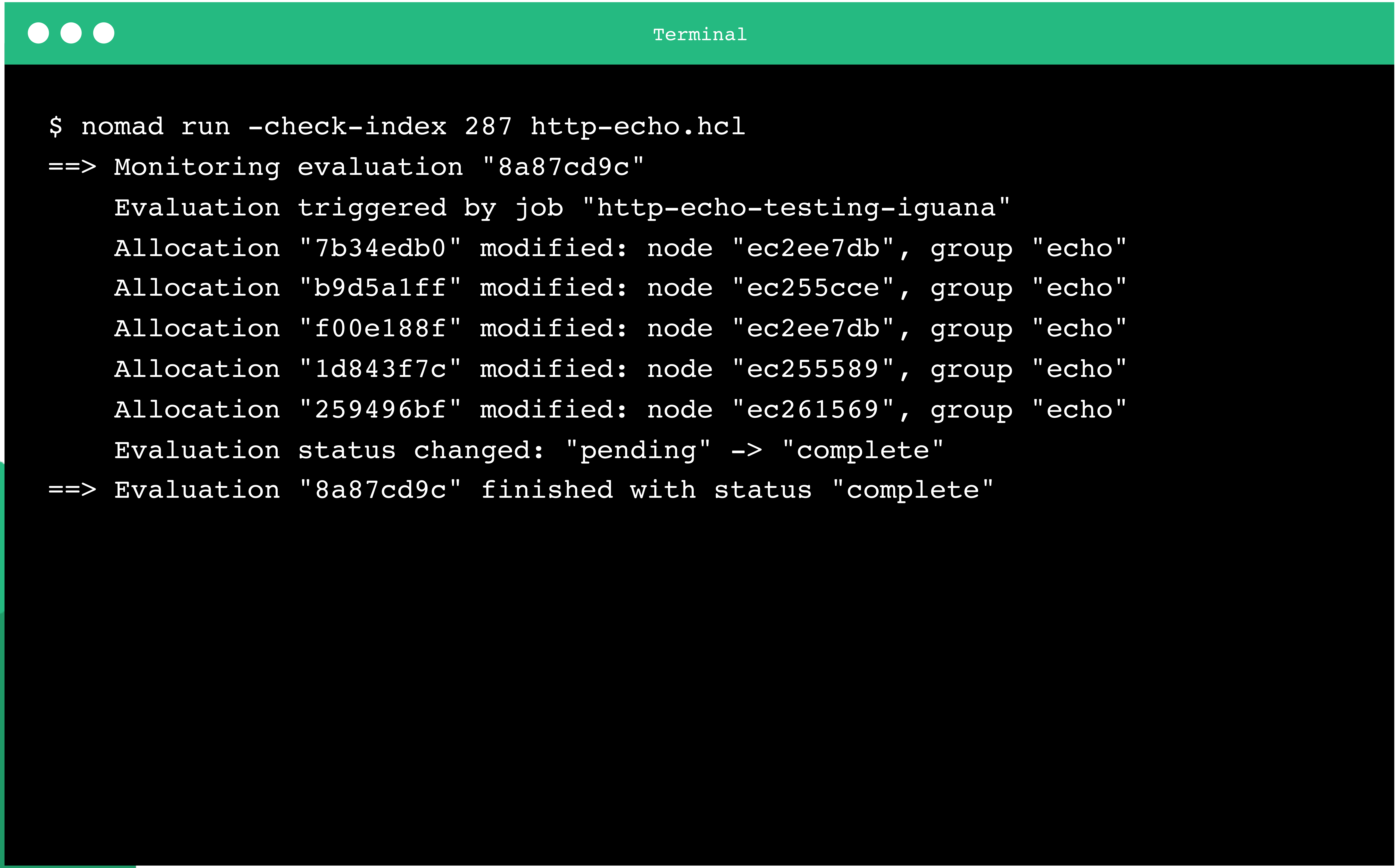
```
$ nomad plan http-echo.nomad
```



```
$ nomad plan http-echo.nomad
+/- Job: "http-echo-testing-iguana"
Task Group: "echo" (5 in-place update)
  Task: "server"

Scheduler dry-run:
- All tasks successfully allocated.

Job Modify Index: 287
To submit the job with version verification run:

nomad run -check-index 287 http-echo.nomad

When running the job with the check-index flag, the job will only be run if
the
server side version matches the job modify index returned. If the index has
changed, another user has modified the job and the plan's results are
potentially invalid.
```



```
$ nomad run -check-index 287 http-echo.nomad
```



```
$ nomad run -check-index 287 http-echo.hcl
==> Monitoring evaluation "8a87cd9c"
    Evaluation triggered by job "http-echo-testing-iguana"
    Allocation "7b34edb0" modified: node "ec2ee7db", group "echo"
    Allocation "b9d5a1ff" modified: node "ec255cce", group "echo"
    Allocation "f00e188f" modified: node "ec2ee7db", group "echo"
    Allocation "1d843f7c" modified: node "ec255589", group "echo"
    Allocation "259496bf" modified: node "ec261569", group "echo"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "8a87cd9c" finished with status "complete"
```

About: Constraints

Constraints are requirements the scheduler must evaluate *about the client* such as operating system, architecture, kernel version, etc.

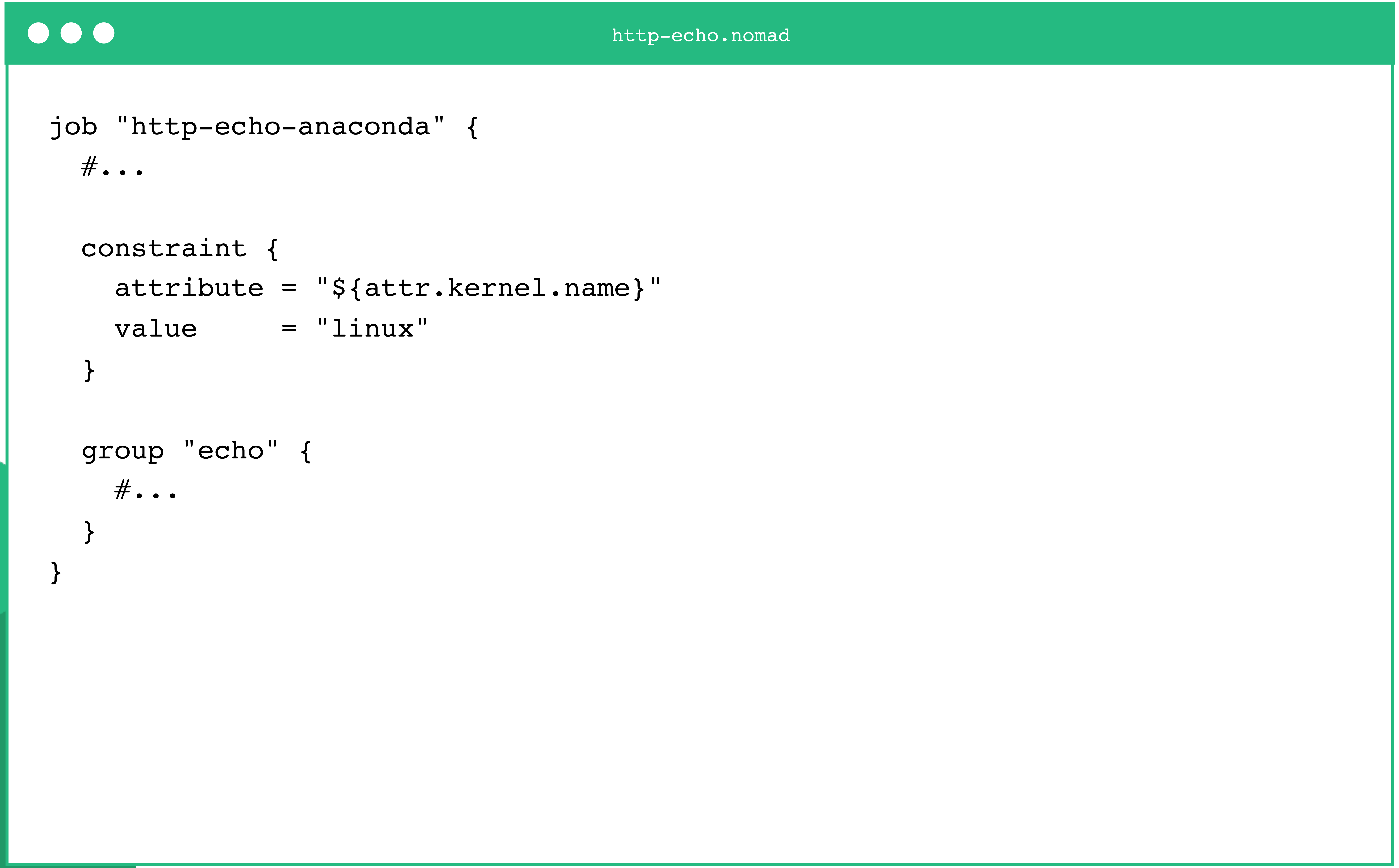
Constraint requirements can be specified at the job, group, or task level.

Exercise: Add Constraint Requirement

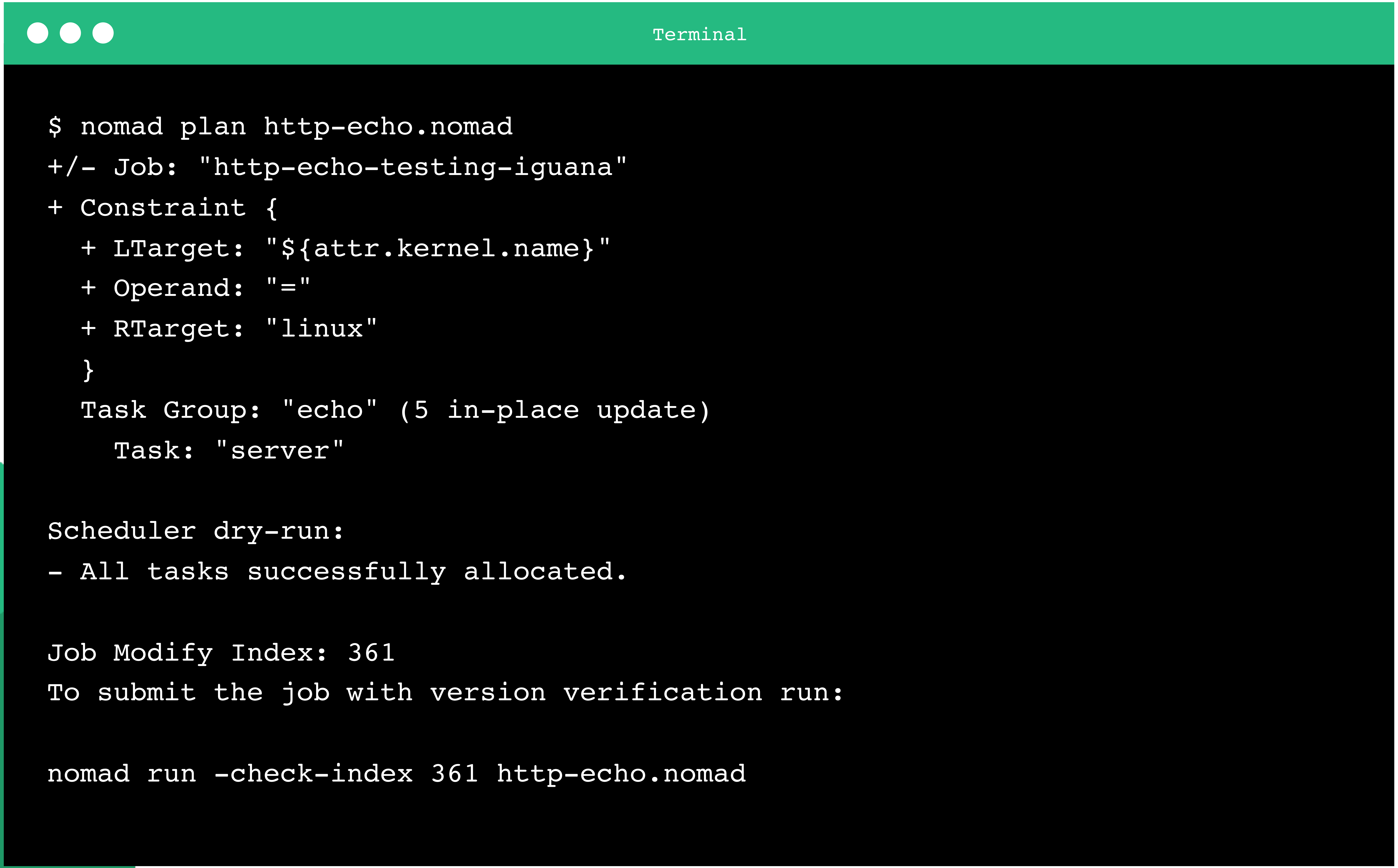
Add a constraint requirement to the `http-echo` job which requires the client kernel to be linux.

Plan this job to see the changes that will take place.

Submit this job to the Nomad server.

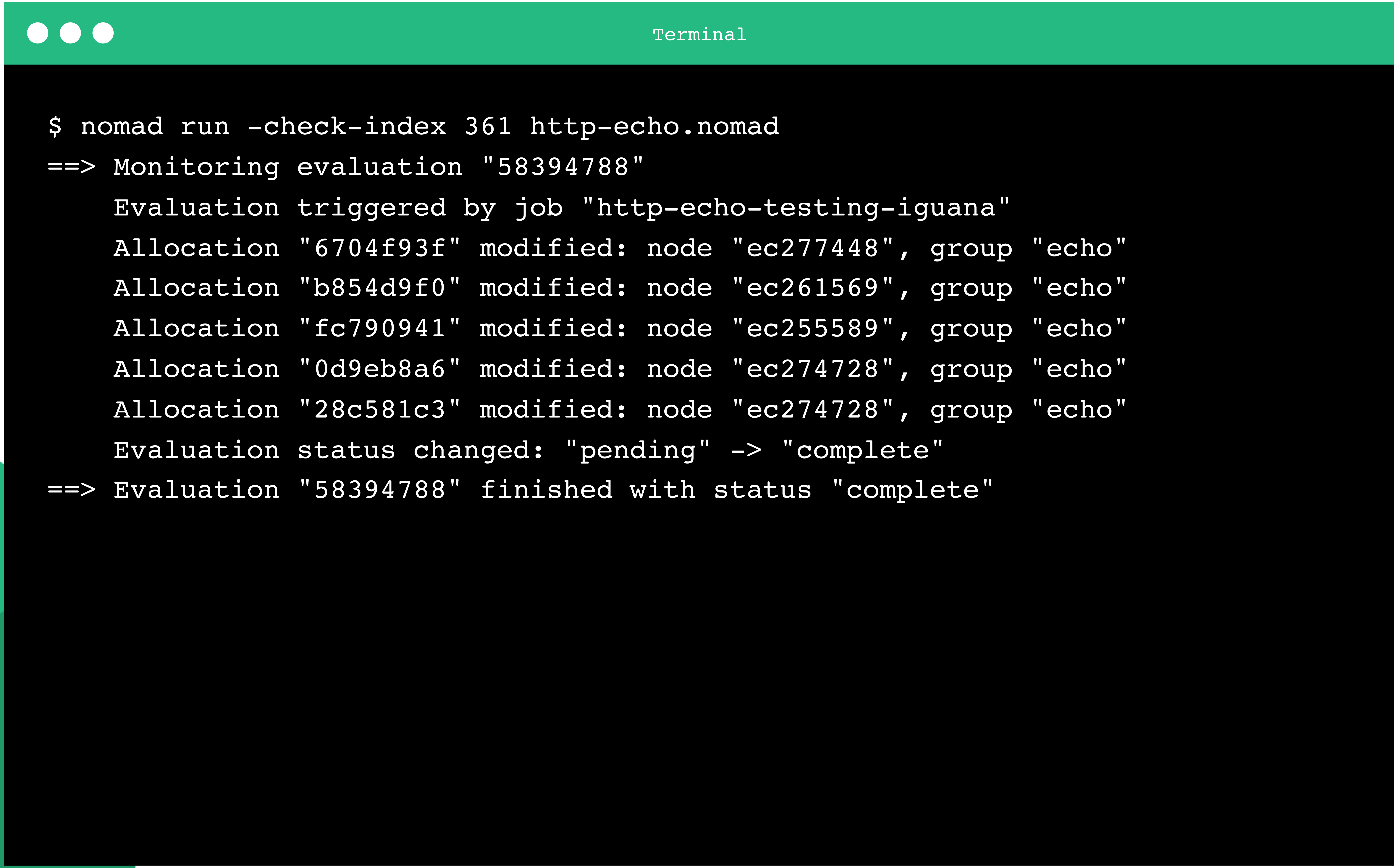


```
job "http-echo-anaconda" {
  #...

  constraint {
    attribute = "${attr.kernel.name}"
    value     = "linux"
  }

  group "echo" {
    #...
  }
}
```



```
$ nomad plan http-echo.nomad
+/- Job: "http-echo-testing-iguana"
+ Constraint {
  + LTarget: "${attr.kernel.name}"
  + Operand: "="
  + RTarget: "linux"
  }
  Task Group: "echo" (5 in-place update)
    Task: "server"

Scheduler dry-run:
- All tasks successfully allocated.

Job Modify Index: 361
To submit the job with version verification run:

nomad run -check-index 361 http-echo.nomad
```

HashiCorp



```
$ nomad run -check-index 361 http-echo.nomad
==> Monitoring evaluation "58394788"
    Evaluation triggered by job "http-echo-testing-iguana"
    Allocation "6704f93f" modified: node "ec277448", group "echo"
    Allocation "b854d9f0" modified: node "ec261569", group "echo"
    Allocation "fc790941" modified: node "ec255589", group "echo"
    Allocation "0d9eb8a6" modified: node "ec274728", group "echo"
    Allocation "28c581c3" modified: node "ec274728", group "echo"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "58394788" finished with status "complete"
```

About: Resources

Resources are minimum requirements the *task must have* to run on the client such as memory or cpu.

Resource constraints can only be specified on the task.

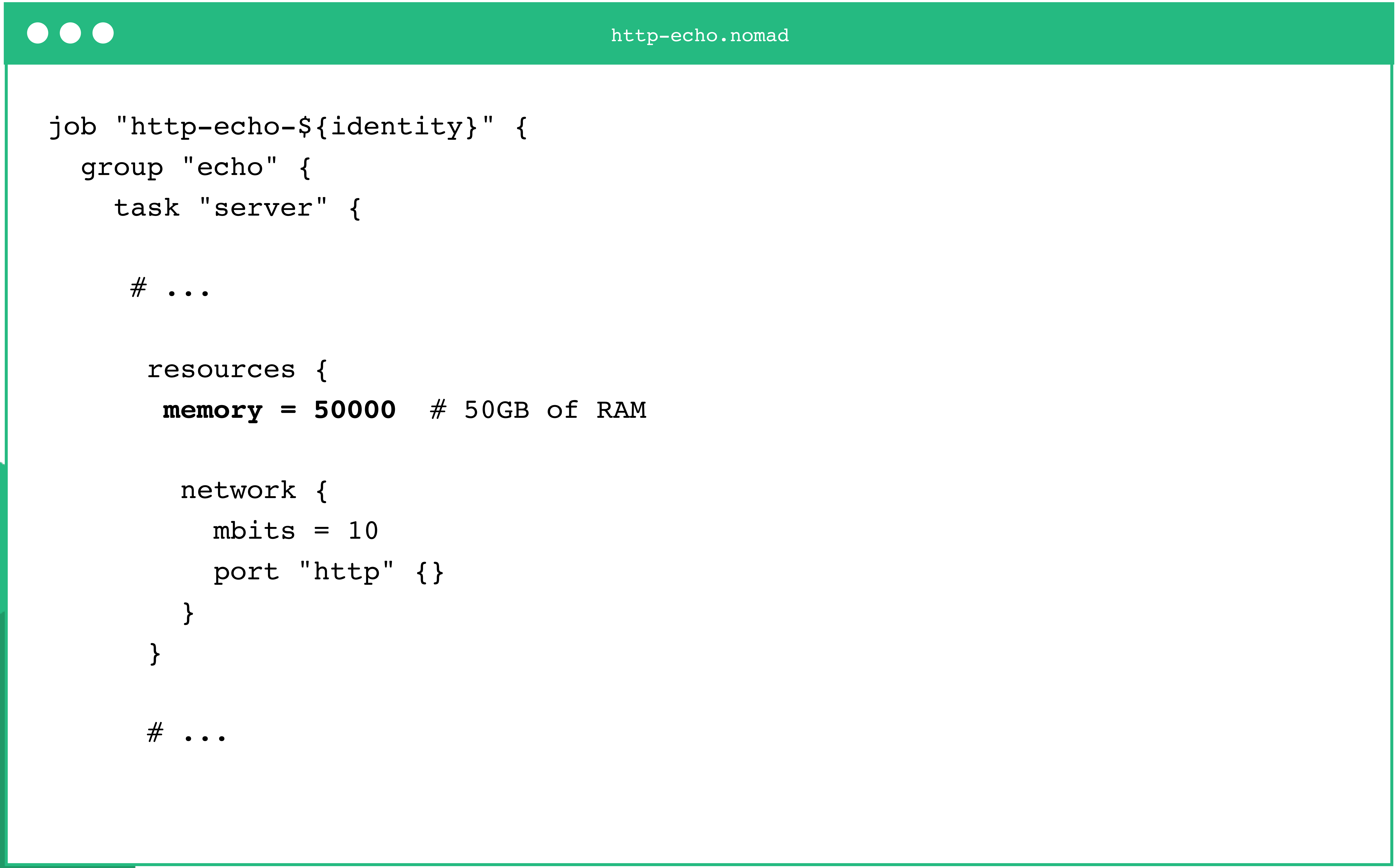
Exercise: Add Resource Requirement

Add a resource requirement to the `http-echo` job on the `server` task which allocates 50GB of memory.

Plan this job to see the changes that will take place.

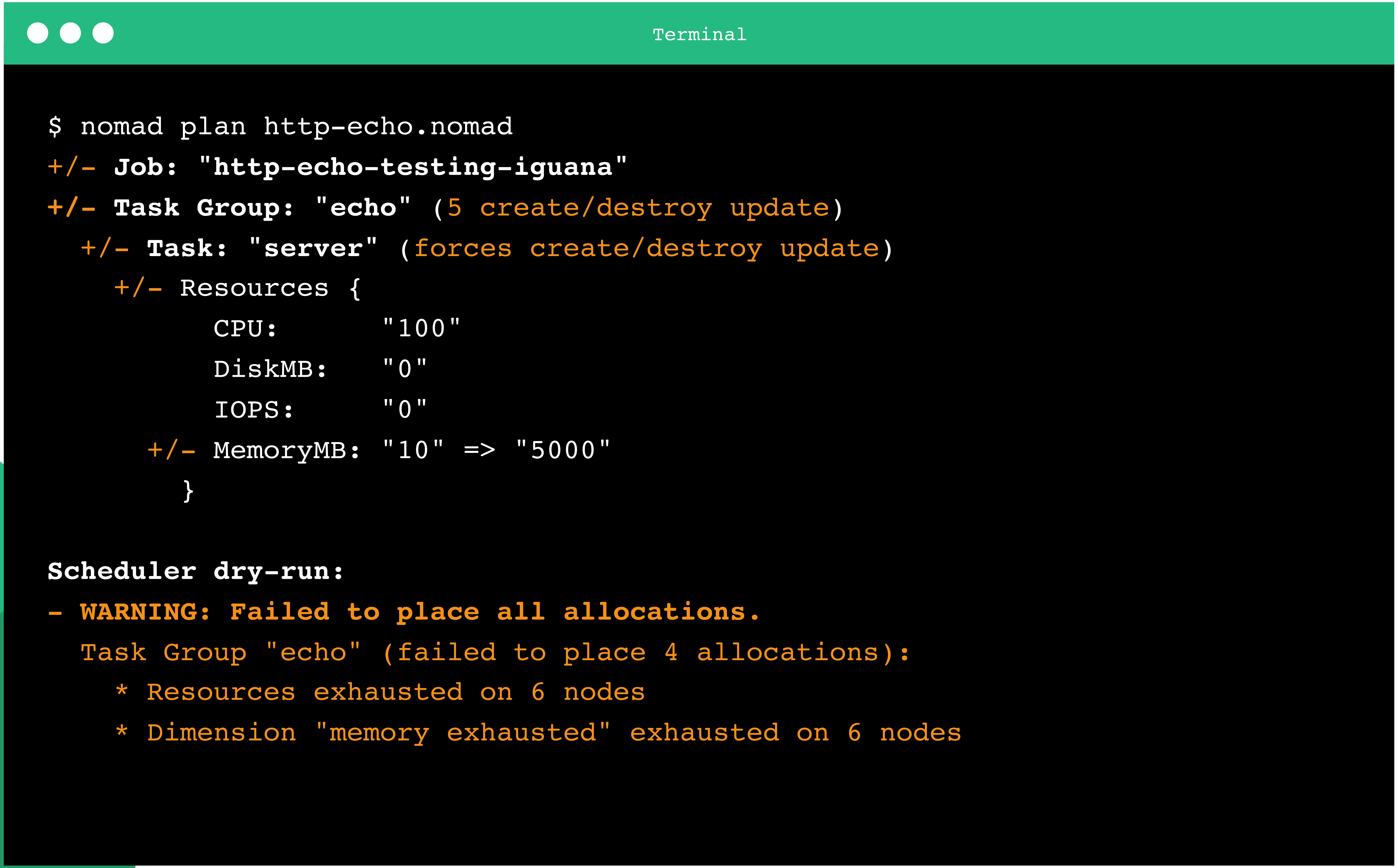
Submit this job to the Nomad server.

Inspect the allocation status.

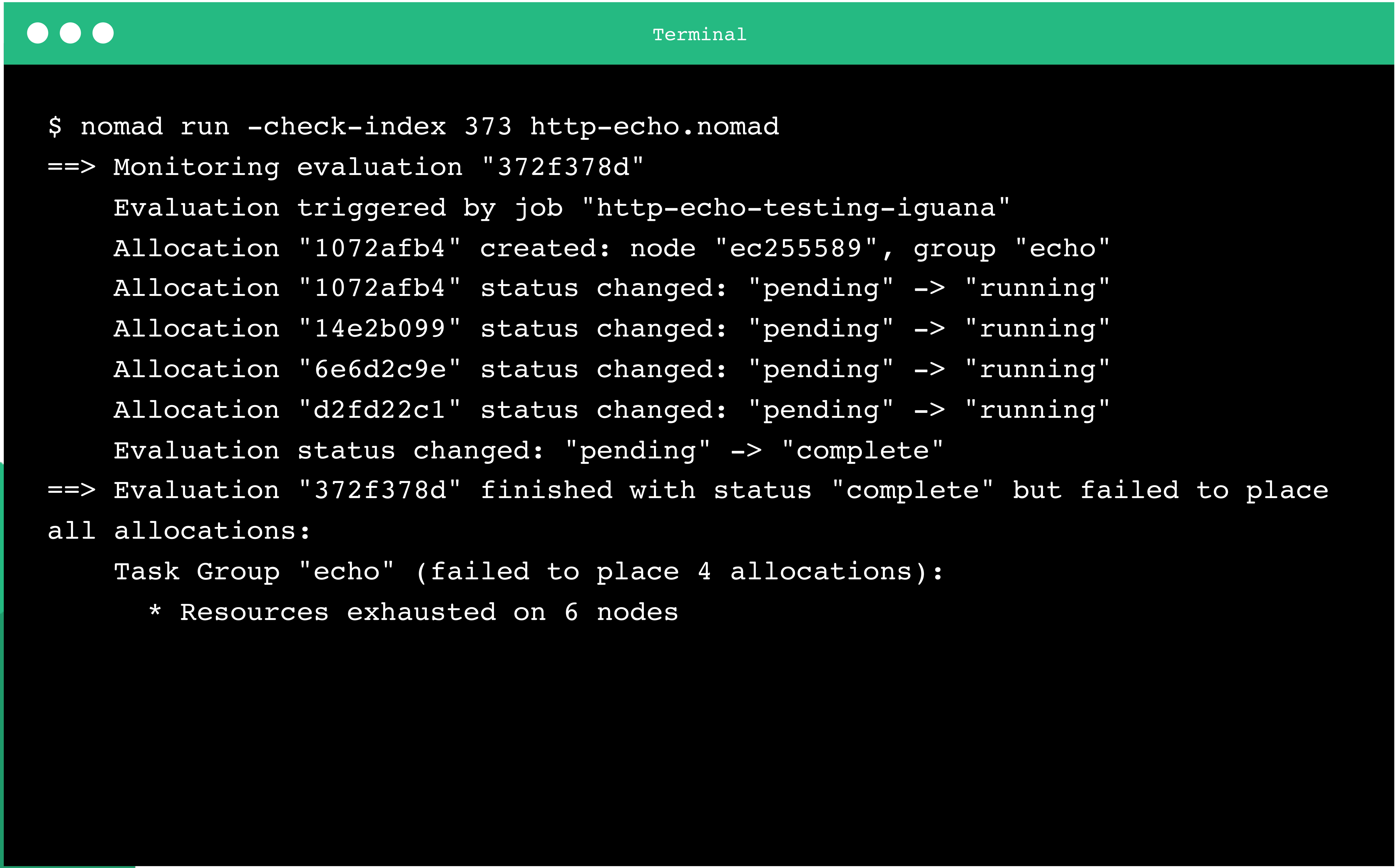


http-echo.nomad

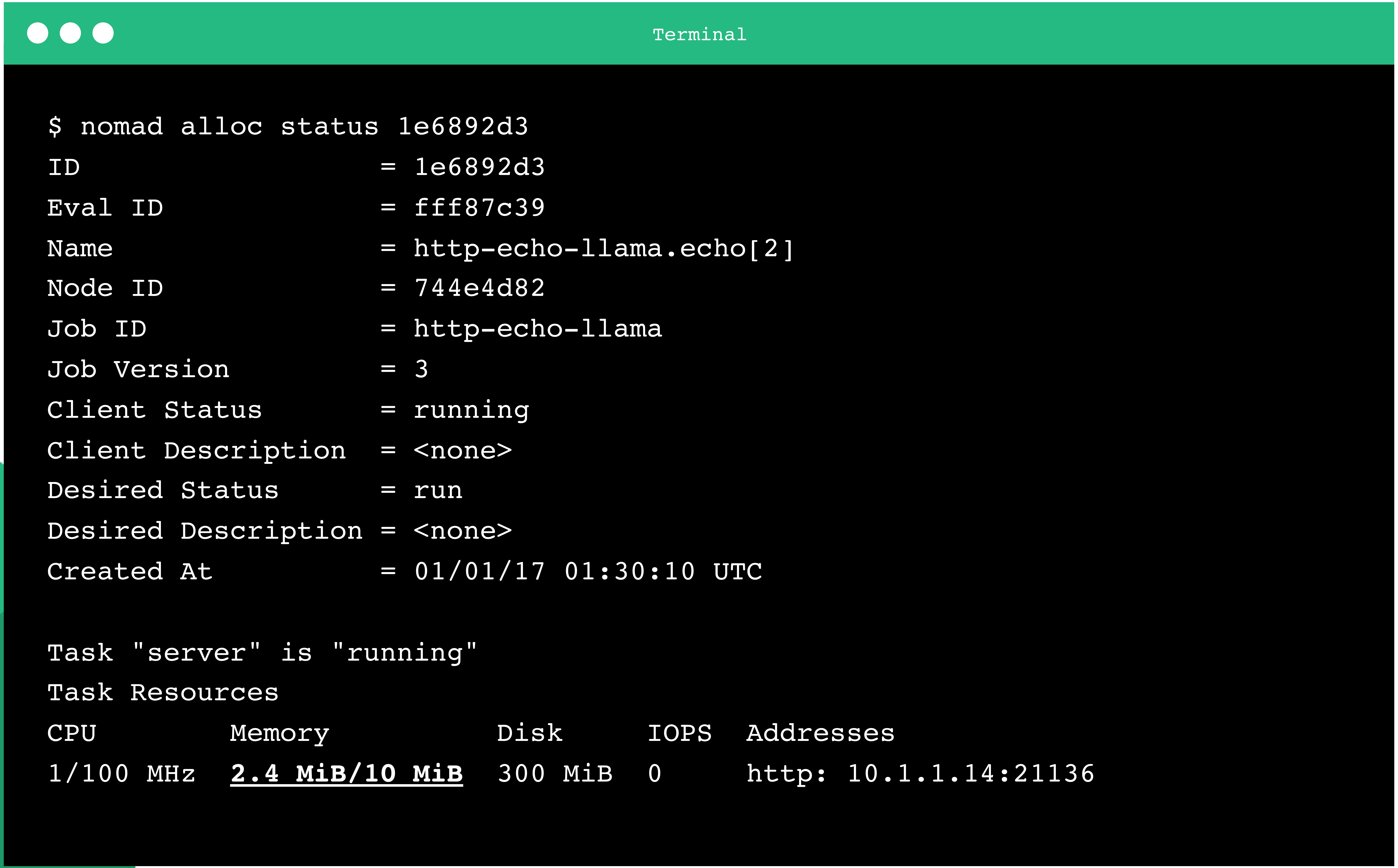
```
job "http-echo-${identity}" {
  group "echo" {
    task "server" {

     # ...

      resources {
       memory = 50000  # 50GB of RAM

        network {
          mbits = 10
          port "http" {}
        }
      }

      # ...
```



```
$ nomad plan http-echo.nomad
+/- Job: "http-echo-testing-iguana"
+/- Task Group: "echo" (5 create/destroy update)
 +/- Task: "server" (forces create/destroy update)
  +/- Resources {
     CPU:      "100"
     DiskMB:   "0"
     IOPS:     "0"
   +/- MemoryMB: "10" => "5000"
    }

Scheduler dry-run:
- WARNING: Failed to place all allocations.
 Task Group "echo" (failed to place 4 allocations):
  * Resources exhausted on 6 nodes
  * Dimension "memory exhausted" exhausted on 6 nodes
```



```
$ nomad run -check-index 373 http-echo.nomad
==> Monitoring evaluation "372f378d"
    Evaluation triggered by job "http-echo-testing-iguana"
    Allocation "1072afb4" created: node "ec255589", group "echo"
    Allocation "1072afb4" status changed: "pending" -> "running"
    Allocation "14e2b099" status changed: "pending" -> "running"
    Allocation "6e6d2c9e" status changed: "pending" -> "running"
    Allocation "d2fd22c1" status changed: "pending" -> "running"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "372f378d" finished with status "complete" but failed to place
all allocations:
    Task Group "echo" (failed to place 4 allocations):
      * Resources exhausted on 6 nodes
```



```
$ nomad alloc status 1e6892d3
ID                  = 1e6892d3
Eval ID             = fff87c39
Name                = http-echo-llama.echo[2]
Node ID             = 744e4d82
Job ID              = http-echo-llama
Job Version         = 3
Client Status       = running
Client Description  = <none>
Desired Status      = run
Desired Description = <none>
Created At          = 01/01/17 01:30:10 UTC

Task "server" is "running"
Task Resources
CPU        Memory          Disk     IOPS  Addresses
1/100 MHz  2.4 MiB/10 MiB  300 MiB  0     http: 10.1.1.14:21136
```

Revert Changes

Update the `http-echo.nomad` file to use the default resource constraints.

Plan and submit the job.

Service Discovery

Service Discovery

Provide a unified mechanism for addressing services in a microservices-oriented architecture.

Various techniques exist, but DNS is usually easiest as it requires zero-touch integration.

Recall: Previously

Previously our `http-echo` job was hard-bound to port 80, and we changed that to allow for scale.

We used `nomad alloc status` to "cheat" and see the port Nomad chose.

Service discovery is a better solution for identifying and addressing these microservices as they **move throughout the system**.

Move Throughout the System?

As jobs are scheduled, their host and port are unpredictable.

Moreover, as hosts join and leave the cluster, jobs may move throughout the cluster.

Service discovery will adapt to the moving jobs over time, without human intervention.

About: Consul

Consul is a free and open-source tool by HashiCorp that implements service discovery.

It uses the RAFT and gossip protocols to reach massive scale.

It has integrations with health checks, so unhealthy services are not added to the service discovery layer.

Similar client-server model to Nomad.

About: Consul

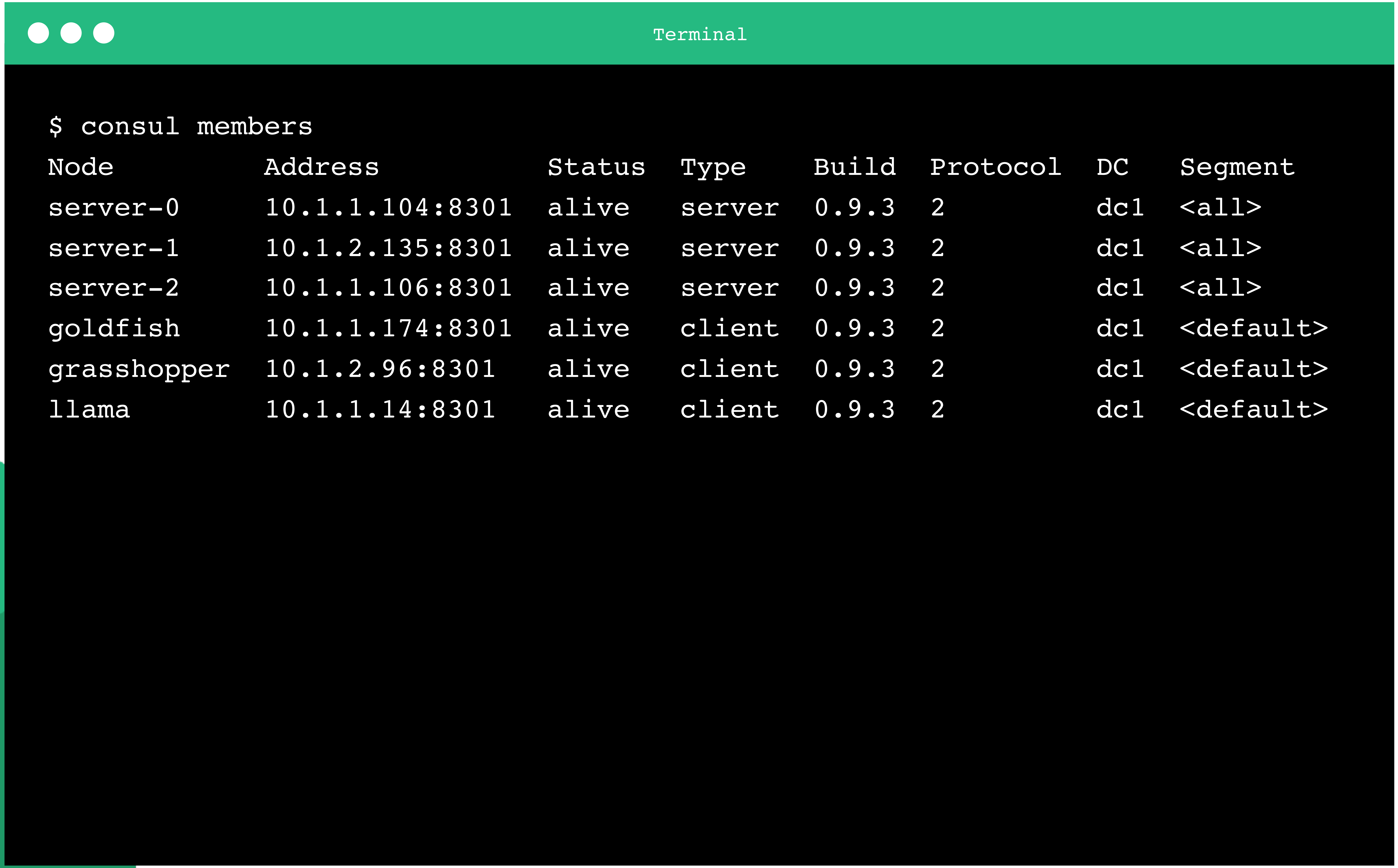
Even though this is not a Consul course, service discovery is a key component of a scheduled architecture.

Consul is already configured and running on your workstation on `127.0.0.1:8500`.

All Consul queries go through the local agent (do not query the service directly).

Exercise: Run `consul members`

Execute the `consul members` command to list all the cluster members.



```
$ consul members
Node         Address          Status  Type    Build  Protocol  DC   Segment
server-0     10.1.1.104:8301  alive   server  0.9.3  2         dc1  <all>
server-1     10.1.2.135:8301  alive   server  0.9.3  2         dc1  <all>
server-2     10.1.1.106:8301  alive   server  0.9.3  2         dc1  <all>
goldfish     10.1.1.174:8301  alive   client  0.9.3  2         dc1  <default>
grasshopper  10.1.2.96:8301   alive   client  0.9.3  2         dc1  <default>
llama        10.1.1.14:8301   alive   client  0.9.3  2         dc1  <default>
```

About: Consul DNS Service Discovery

Randomized round-robin to all services that match the query.

Filters based on health checks (unhealthy hosts are not returned from the query).

"Health" is determined by the application.

About: Consul DNS Service Discovery

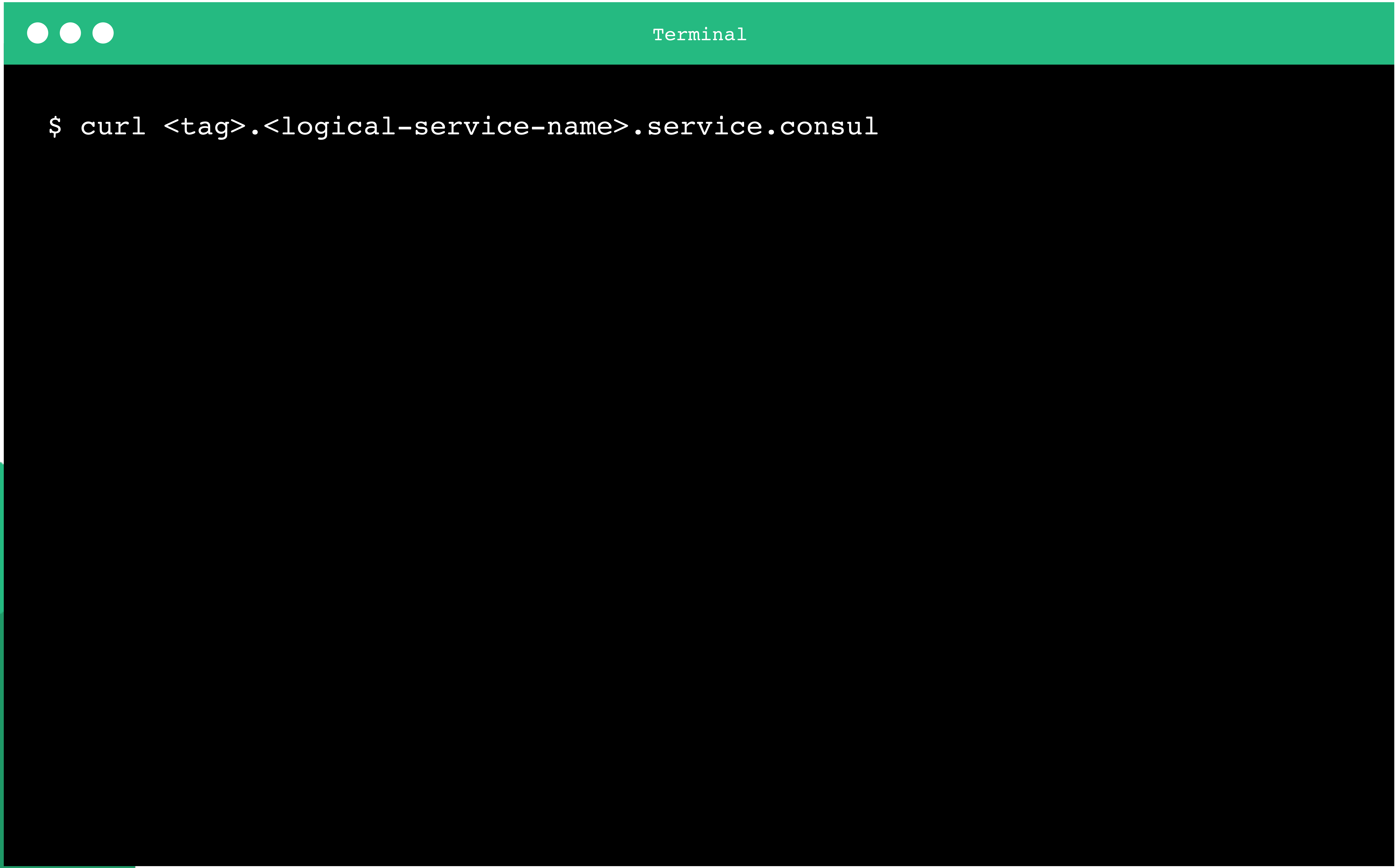
Previously we queried `$(identity).http-echo.service.consul`.

`$(identity)` is a tag (services can have zero or more tags).

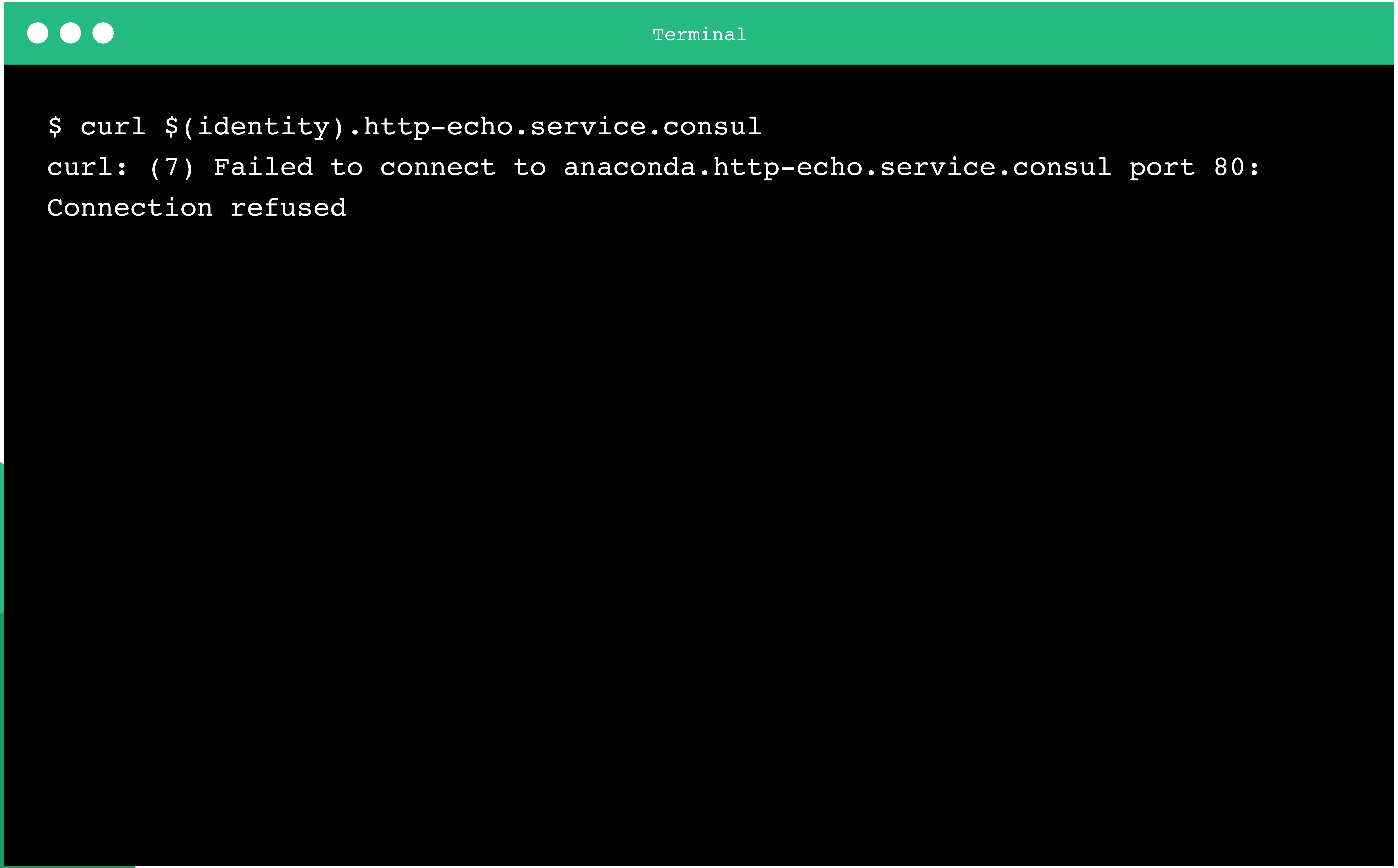
`http-echo` is the logical service name (could have more than one).

`service` is the DNS namespace which queries Consul services.

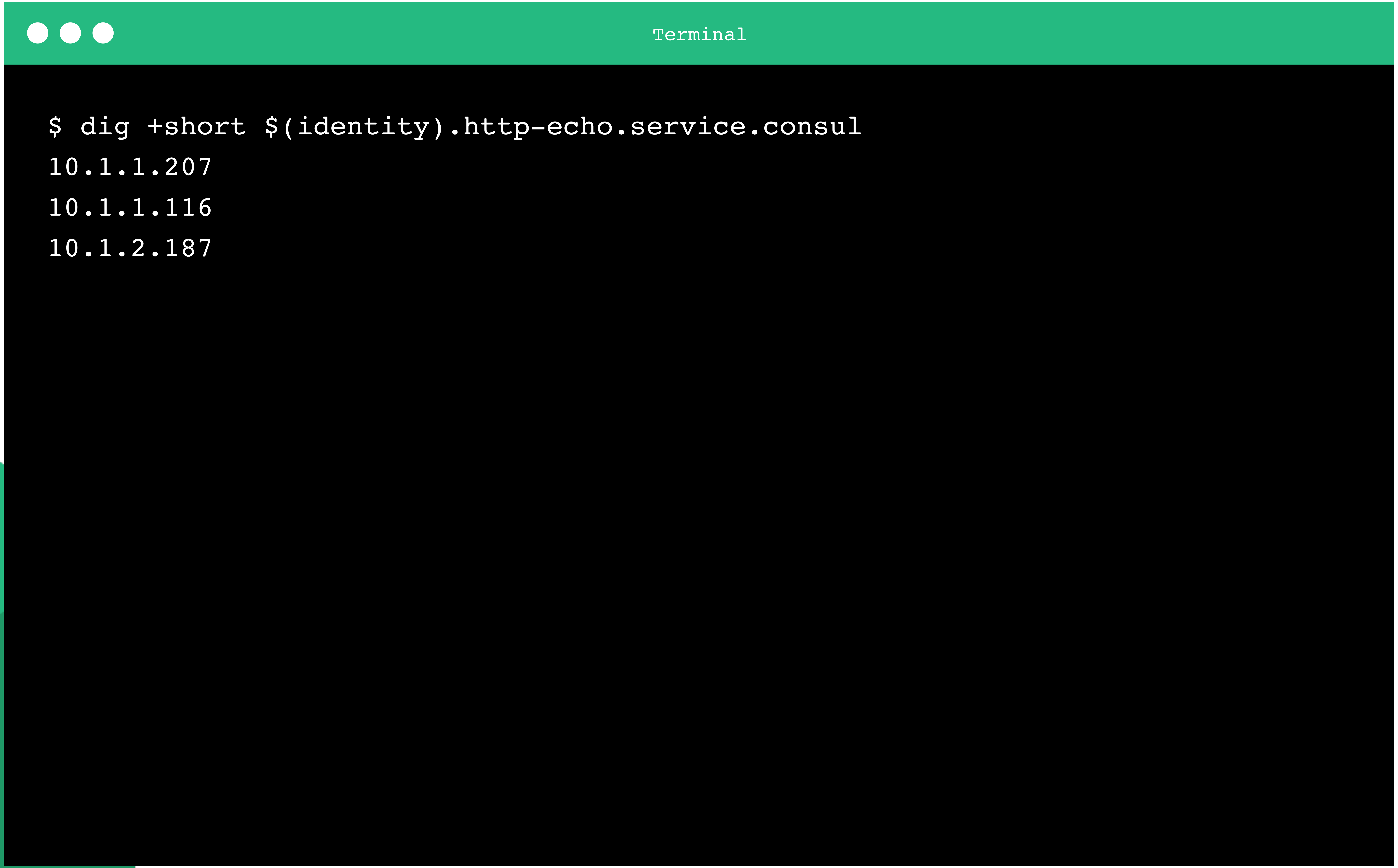
`consul` is the DNS suffix which delegates system DNS to Consul.



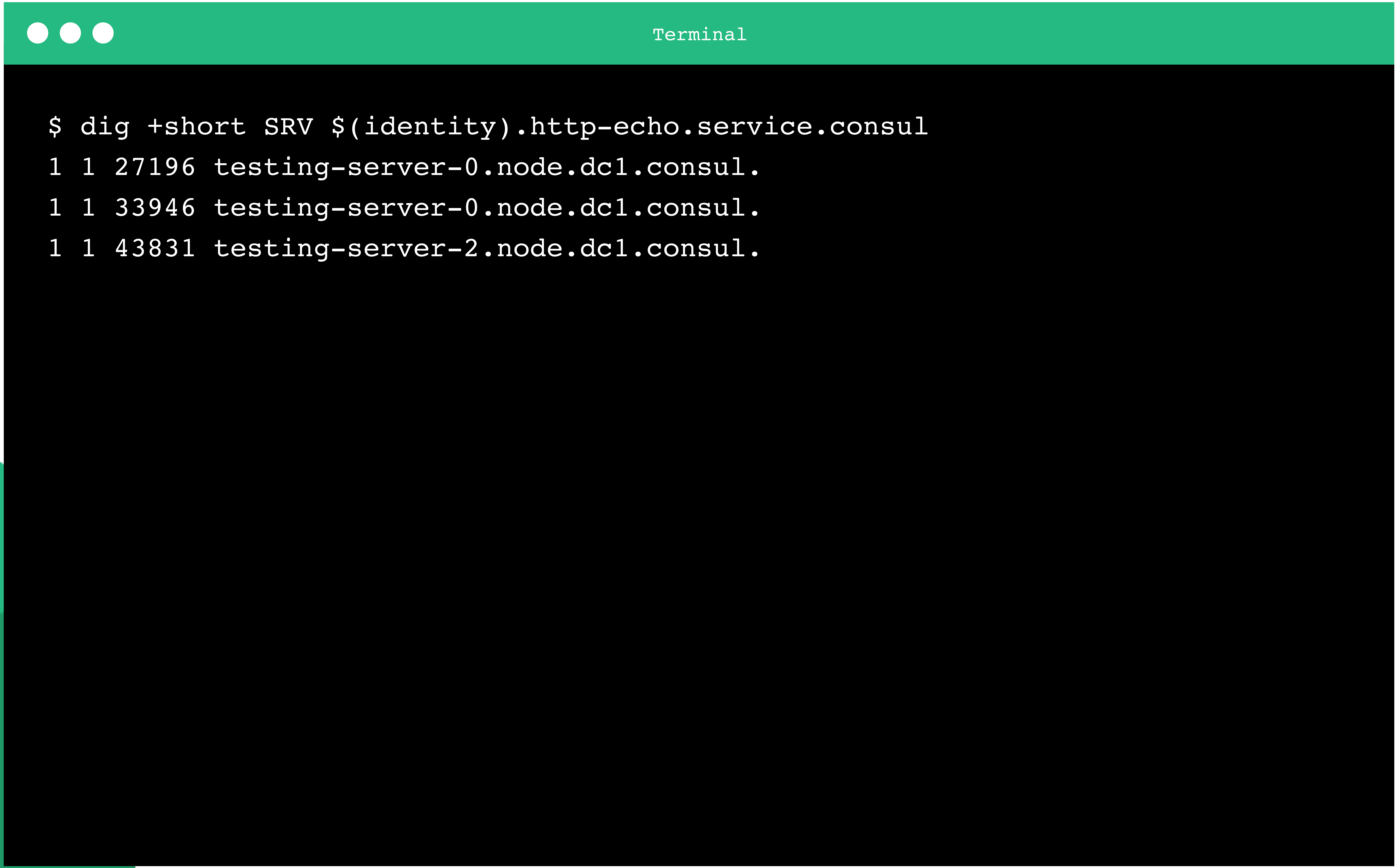
```
$ curl <tag>.<logical-service-name>.service.consul
```



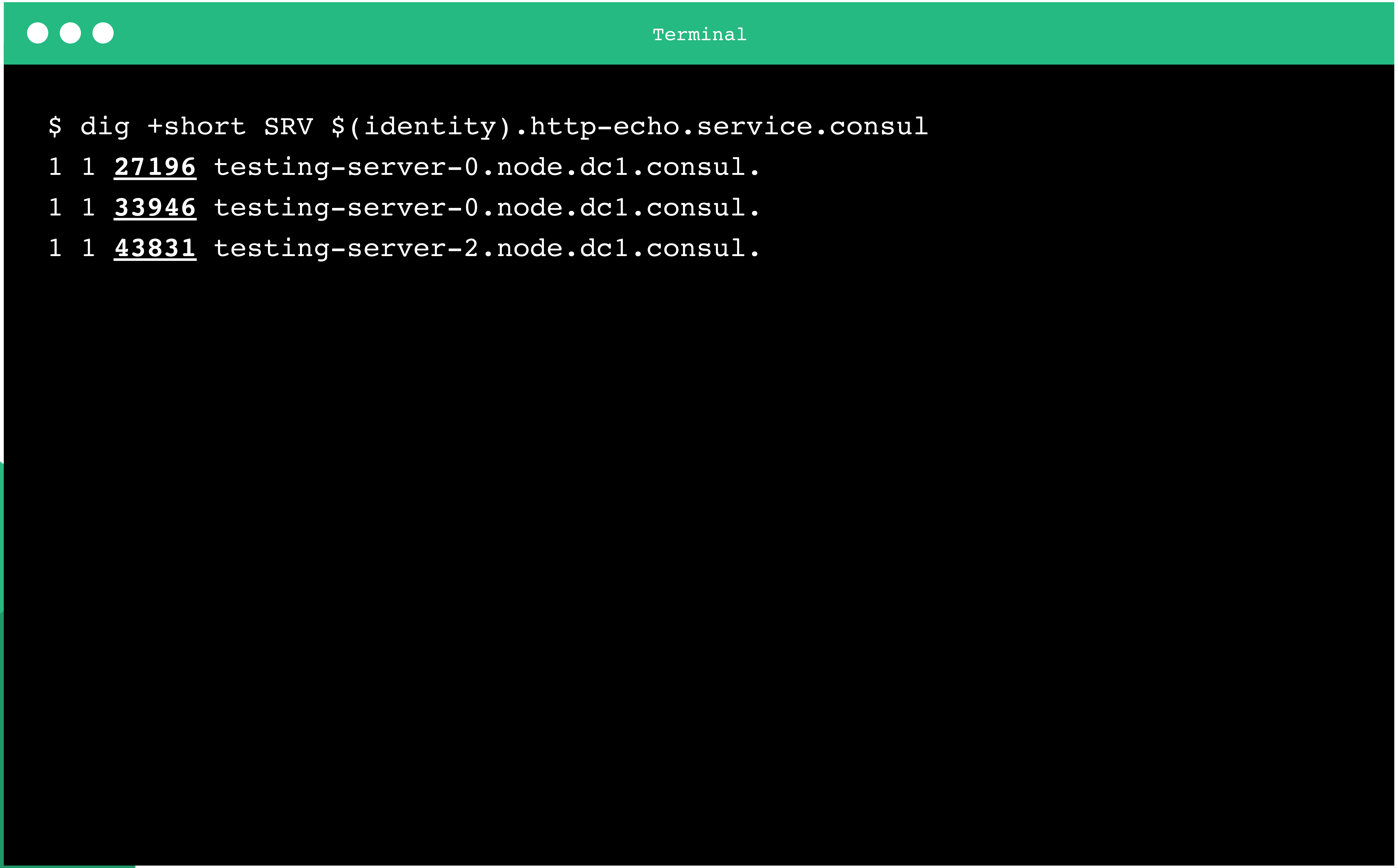
```
$ curl $(identity).http-echo.service.consul
curl: (7) Failed to connect to anaconda.http-echo.service.consul port 80:
Connection refused
```



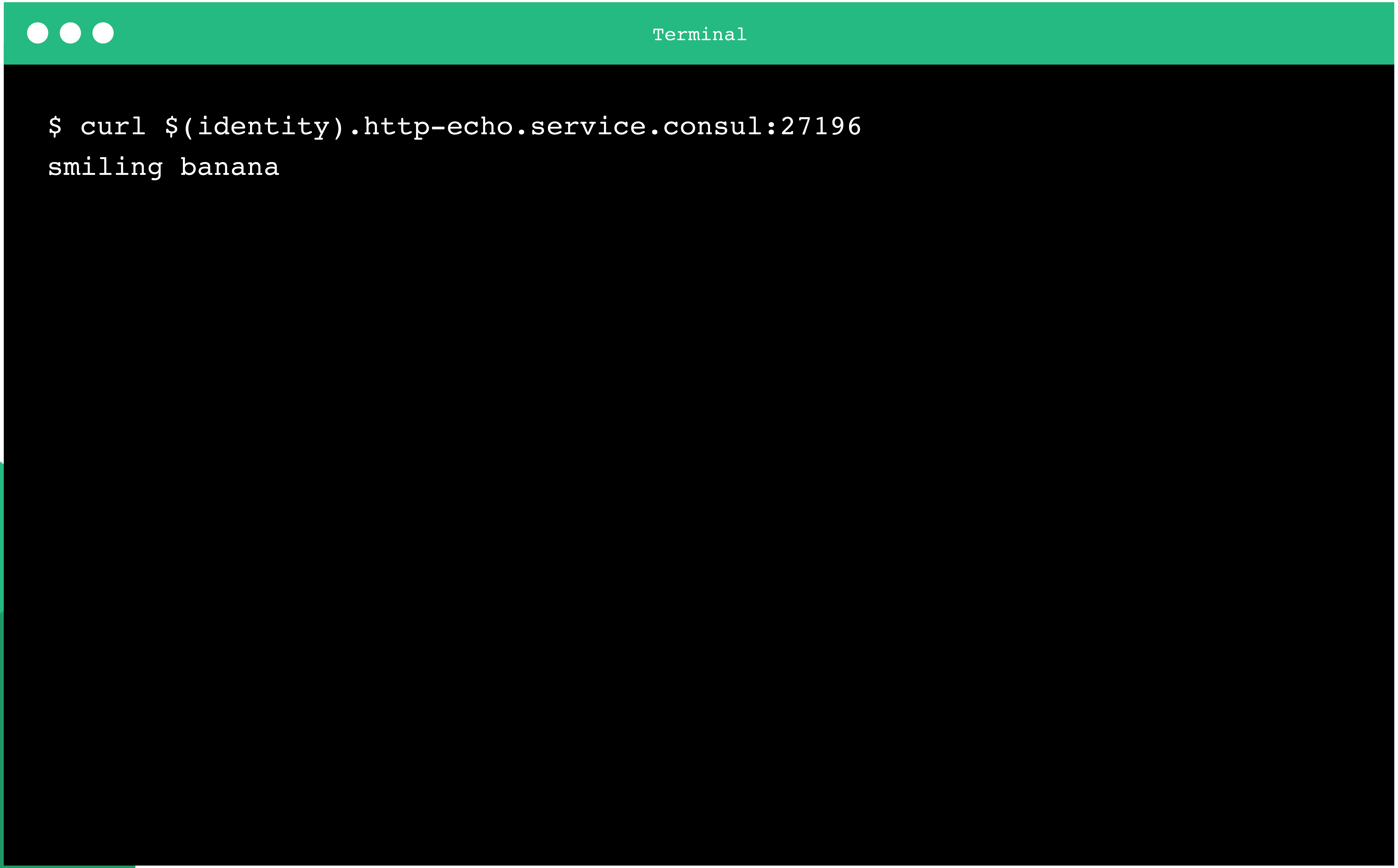
```
$ dig +short $(identity).http-echo.service.consul
10.1.1.207
10.1.1.116
10.1.2.187
```



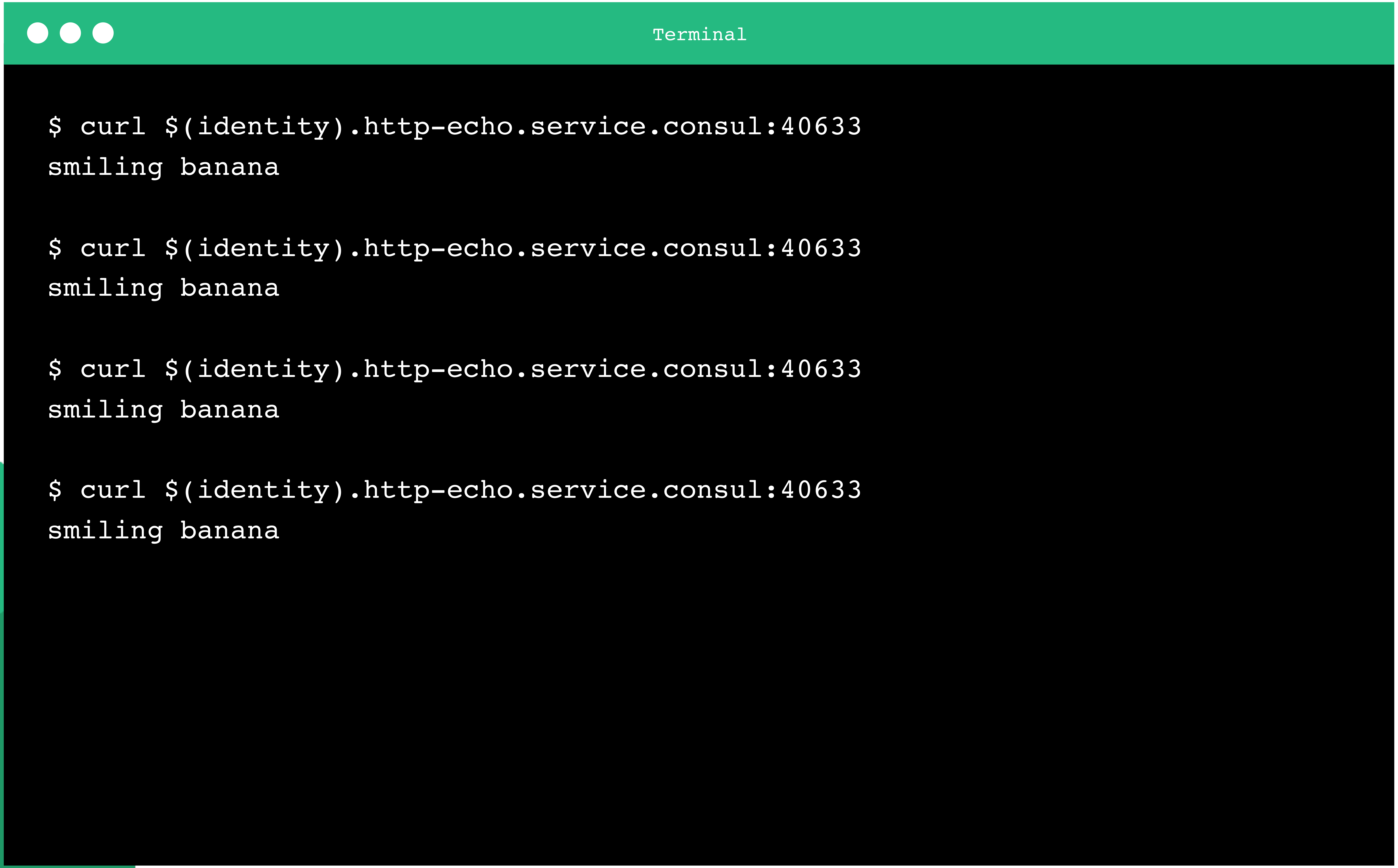
```
$ dig +short SRV $(identity).http-echo.service.consul
1 1 27196 testing-server-0.node.dc1.consul.
1 1 33946 testing-server-0.node.dc1.consul.
1 1 43831 testing-server-2.node.dc1.consul.
```



```
$ dig +short SRV $(identity).http-echo.service.consul
1 1 27196 testing-server-0.node.dc1.consul.
1 1 33946 testing-server-0.node.dc1.consul.
1 1 43831 testing-server-2.node.dc1.consul.
```



```
$ curl $(identity).http-echo.service.consul:27196
smiling banana
```



```
$ curl $(identity).http-echo.service.consul:40633
smiling banana

$ curl $(identity).http-echo.service.consul:40633
smiling banana

$ curl $(identity).http-echo.service.consul:40633
smiling banana

$ curl $(identity).http-echo.service.consul:40633
smiling banana
```

Lessons: Service Discovery

Scheduled architectures make heavy use of service discovery.

Service discovery is integrated with the health of the application.

Nomad uses Consul for service discovery by default.

Load Balancing

Load Balancing

Load balancing is a close cousin of service discovery.

Allows providing a known URL or path to other services.

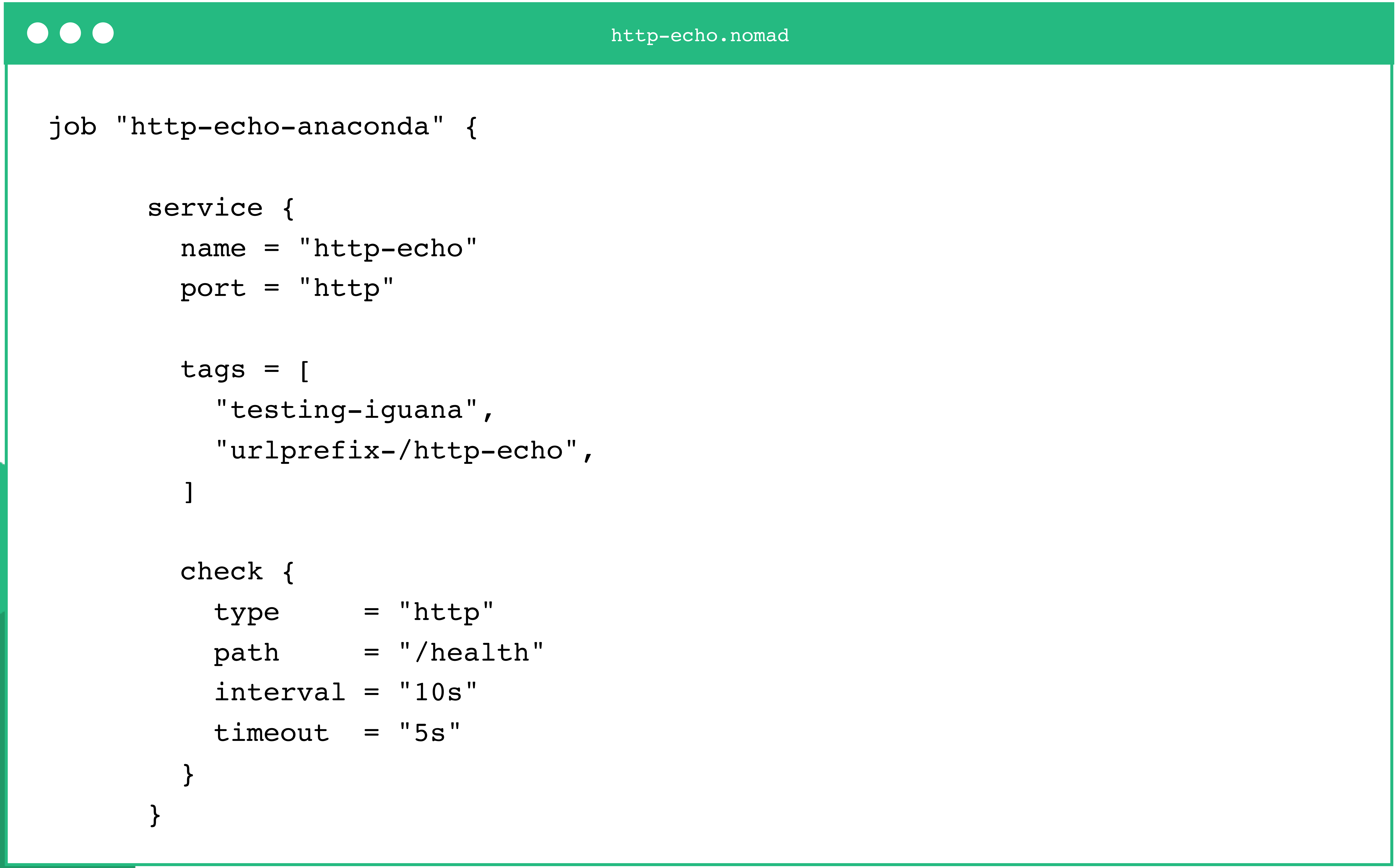
Round robin and integrates health checks.

Load Balancing in Nomad

Load balancing in Nomad is possible through the use of fabio, an open source tool

Fabio integrates with Consul and acts as a load balancer for all healthy services in a given name.

Consul Connect support coming soon!



http-echo.nomad

```
job "http-echo-anaconda" {

      service {
        name = "http-echo"
        port = "http"

        tags = [
          "testing-iguana",
          "urlprefix-/http-echo",
        ]

        check {
          type     = "http"
          path     = "/health"
          interval = "10s"
          timeout  = "5s"
        }
      }
```

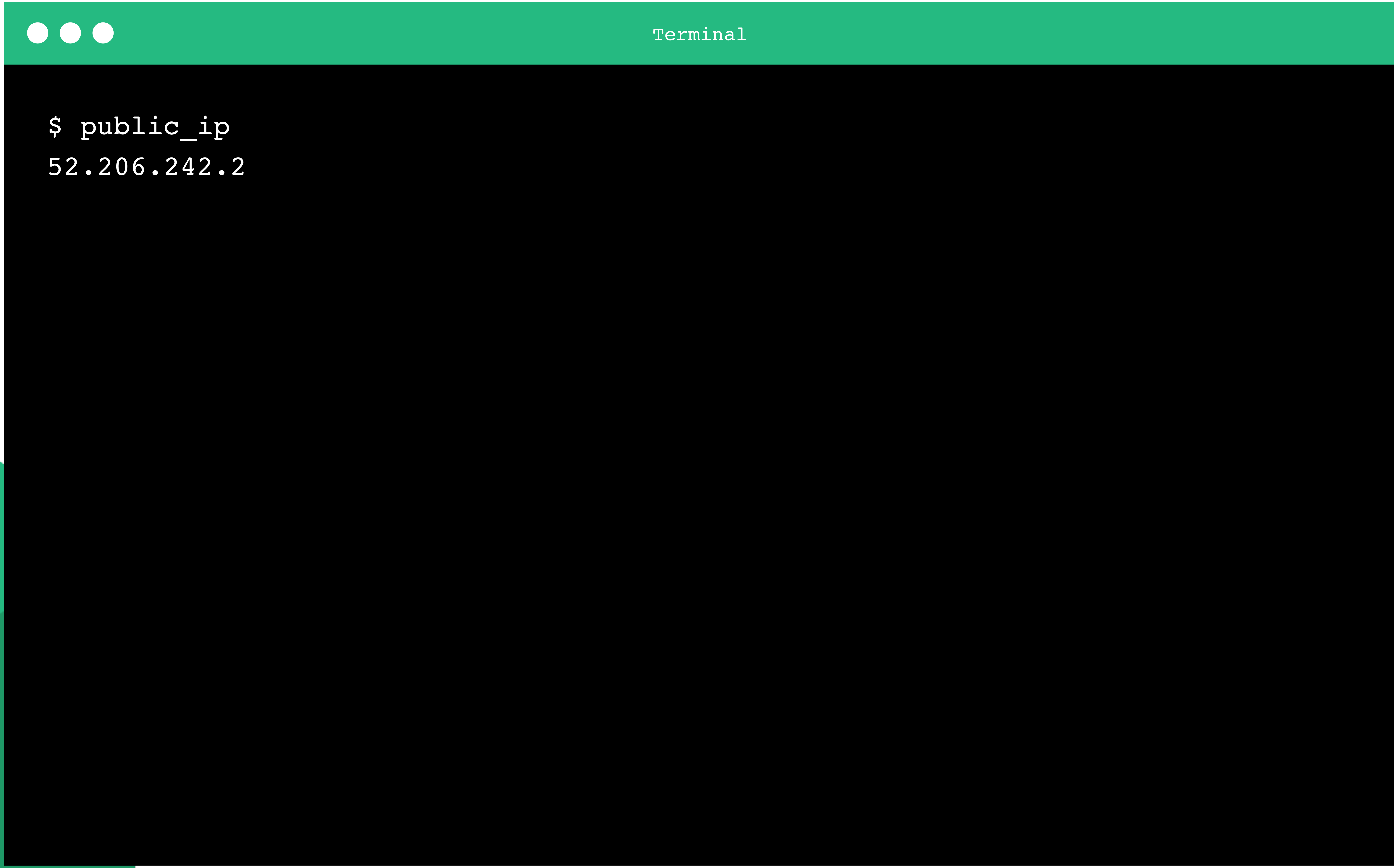
HashiCorp

Exercise: Load Balance

Using your local machine's public IP address, visit the `/http-echo` URL on port 9999 in your browser.

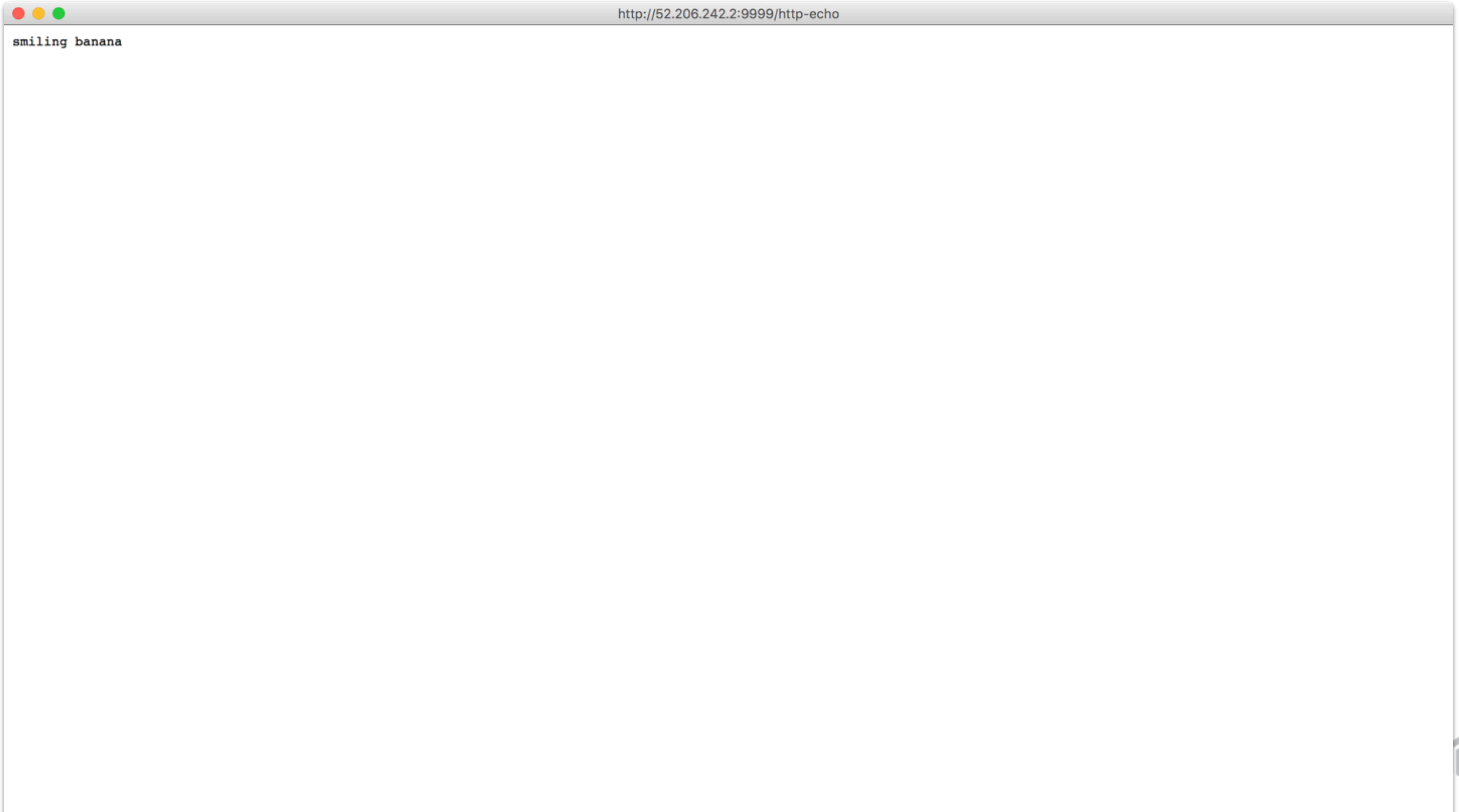
(9999 is the default fabio port)

HINT: Execute `public_ip`



```
$ public_ip
52.206.242.2
```

Visit in Browser



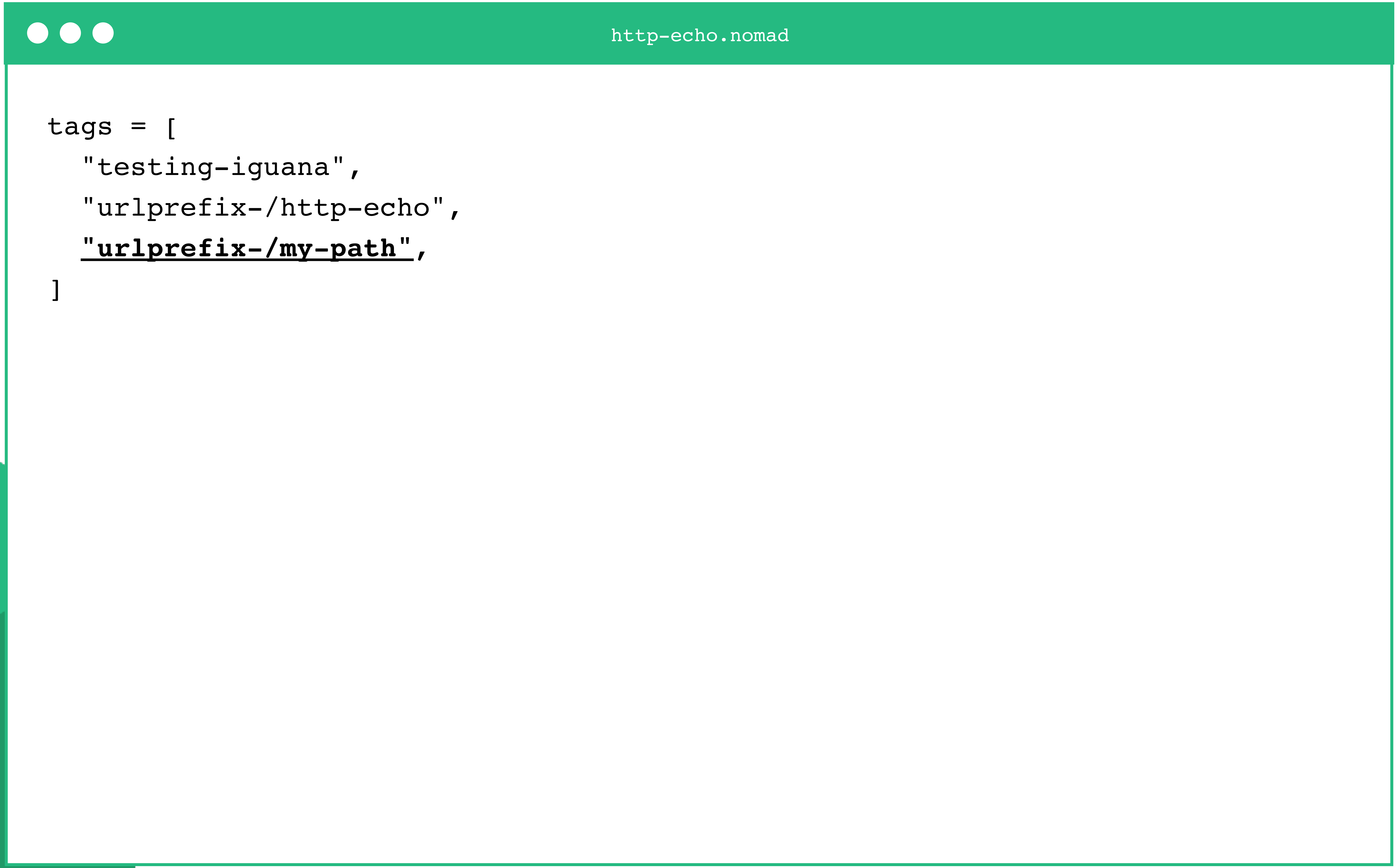
```
smiling banana
```

Exercise: Add URL

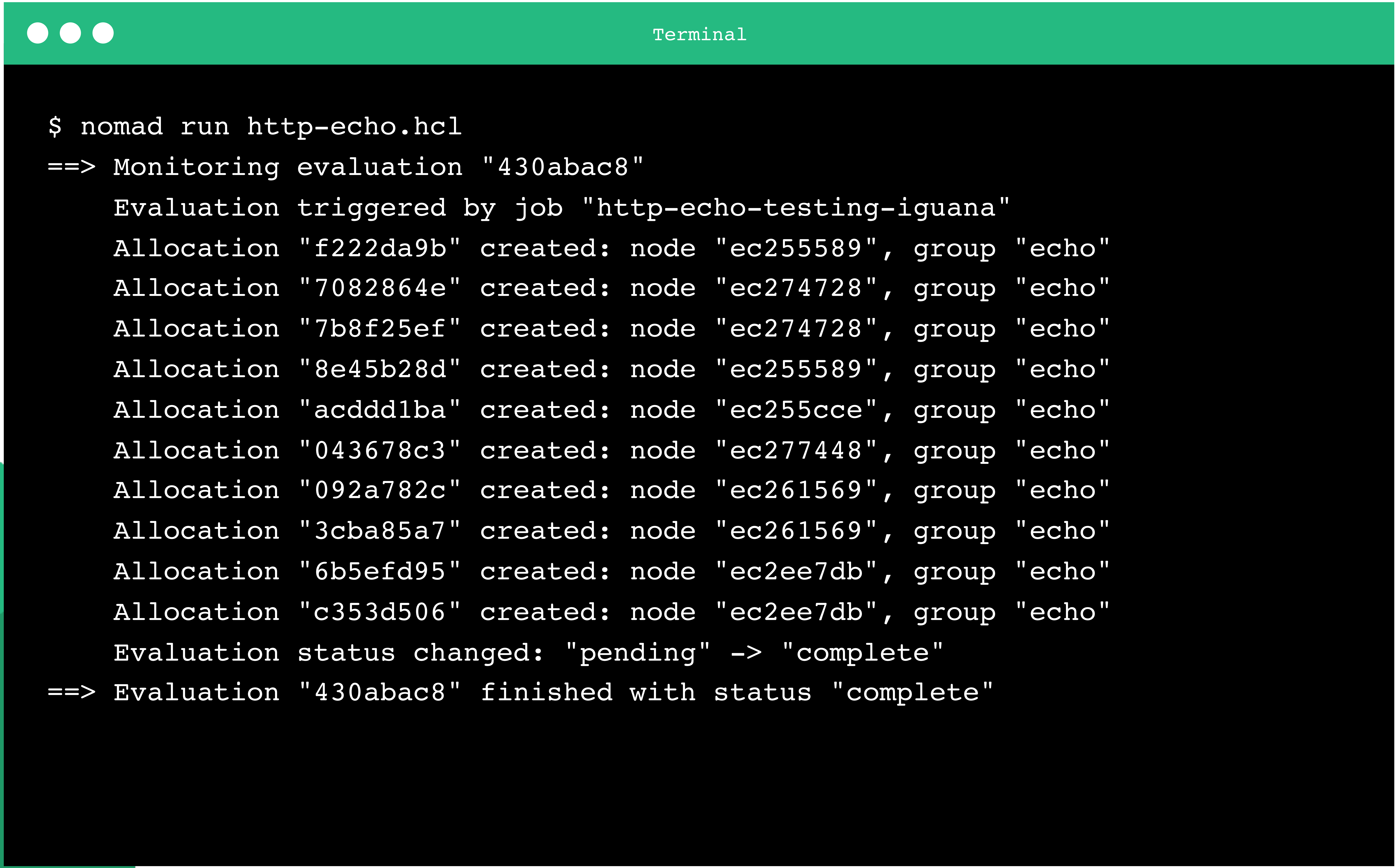
Add an additional URL suffix where your service should be addressable.

Resubmit the nomad job.

Visit the URL in your browser.



```
tags = [
  "testing-iguana",
  "urlprefix-/http-echo",
  "urlprefix-/my-path",
]
```



```
$ nomad run http-echo.hcl
==> Monitoring evaluation "430abac8"
    Evaluation triggered by job "http-echo-testing-iguana"
    Allocation "f222da9b" created: node "ec255589", group "echo"
    Allocation "7082864e" created: node "ec274728", group "echo"
    Allocation "7b8f25ef" created: node "ec274728", group "echo"
    Allocation "8e45b28d" created: node "ec255589", group "echo"
    Allocation "acddd1ba" created: node "ec255cce", group "echo"
    Allocation "043678c3" created: node "ec277448", group "echo"
    Allocation "092a782c" created: node "ec261569", group "echo"
    Allocation "3cba85a7" created: node "ec261569", group "echo"
    Allocation "6b5efd95" created: node "ec2ee7db", group "echo"
    Allocation "c353d506" created: node "ec2ee7db", group "echo"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "430abac8" finished with status "complete"
```

Downloading Artifacts

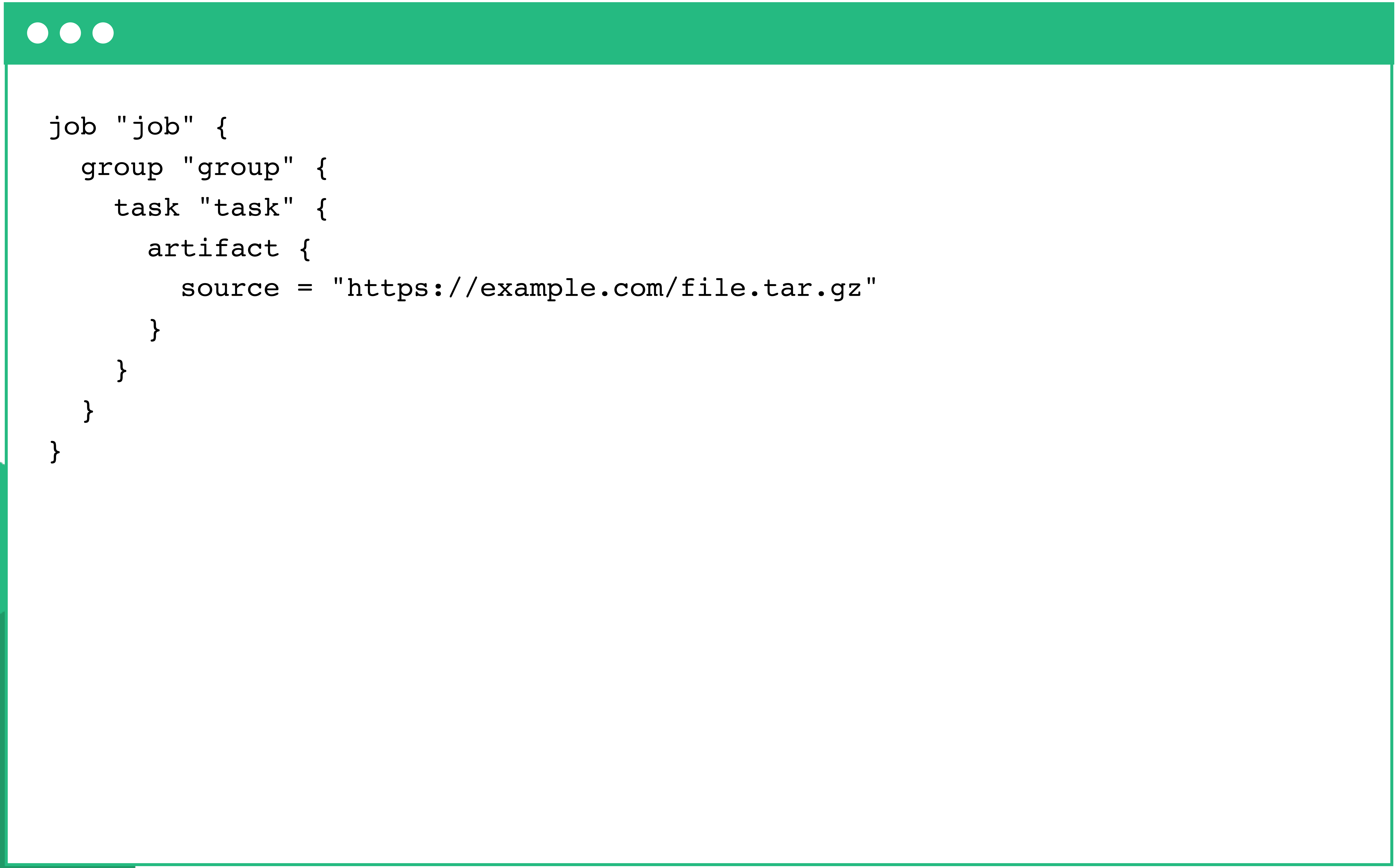
Nomad Artifacts

Many jobs need external files to run such as configuration or the even the binary itself.

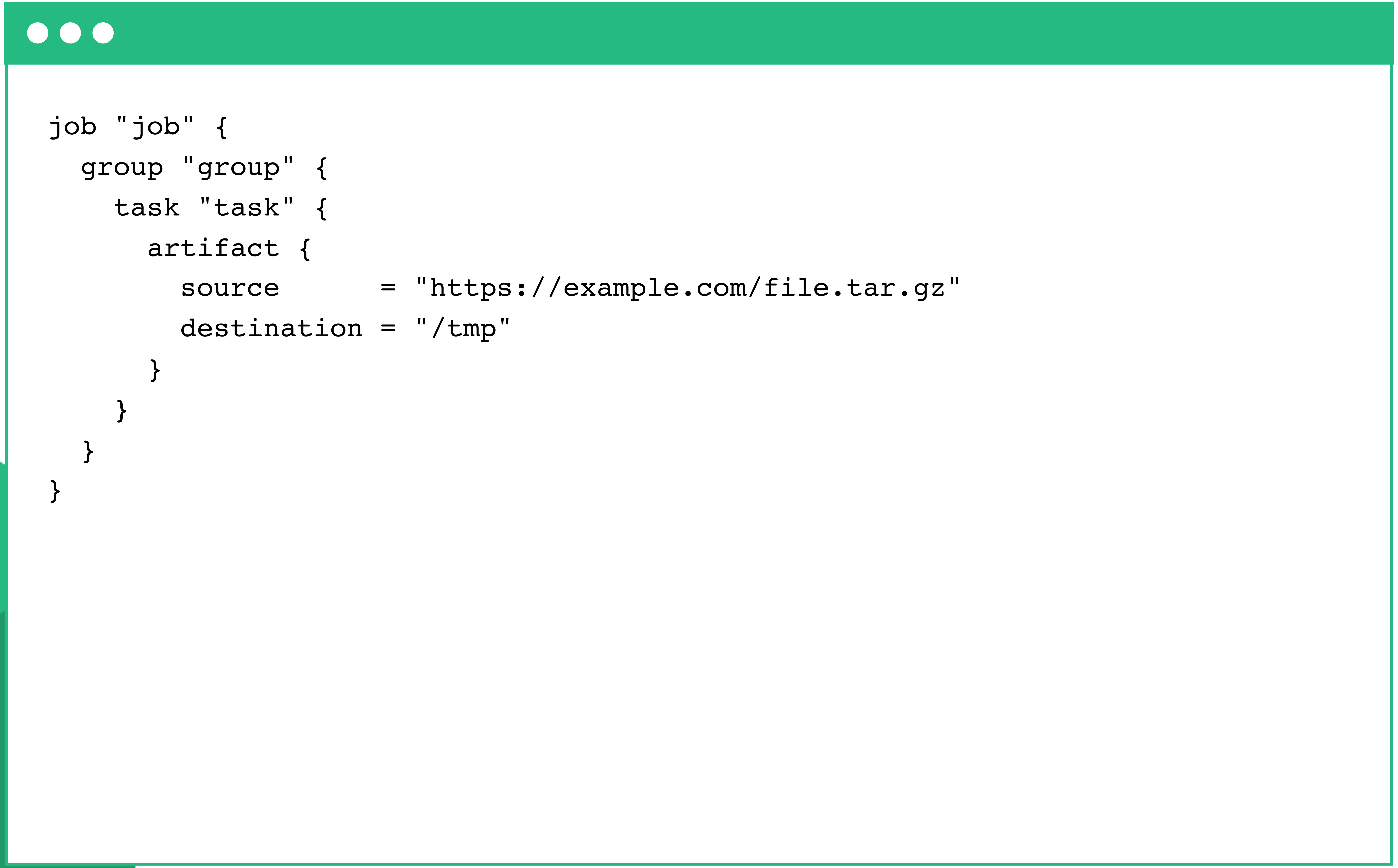
Nomad supports downloading artifacts from many different sources including S3 buckets, git, and more.

Optional checksum verification can ensure integrity.

Automatically extracts commonly-known archive types.



```
job "job" {
  group "group" {
    task "task" {
      artifact {
        source = "https://example.com/file.tar.gz"
      }
    }
  }
}
```



```
job "job" {
  group "group" {
    task "task" {
      artifact {
        source      = "https://example.com/file.tar.gz"
        destination = "/tmp"
      }
    }
  }
}
```

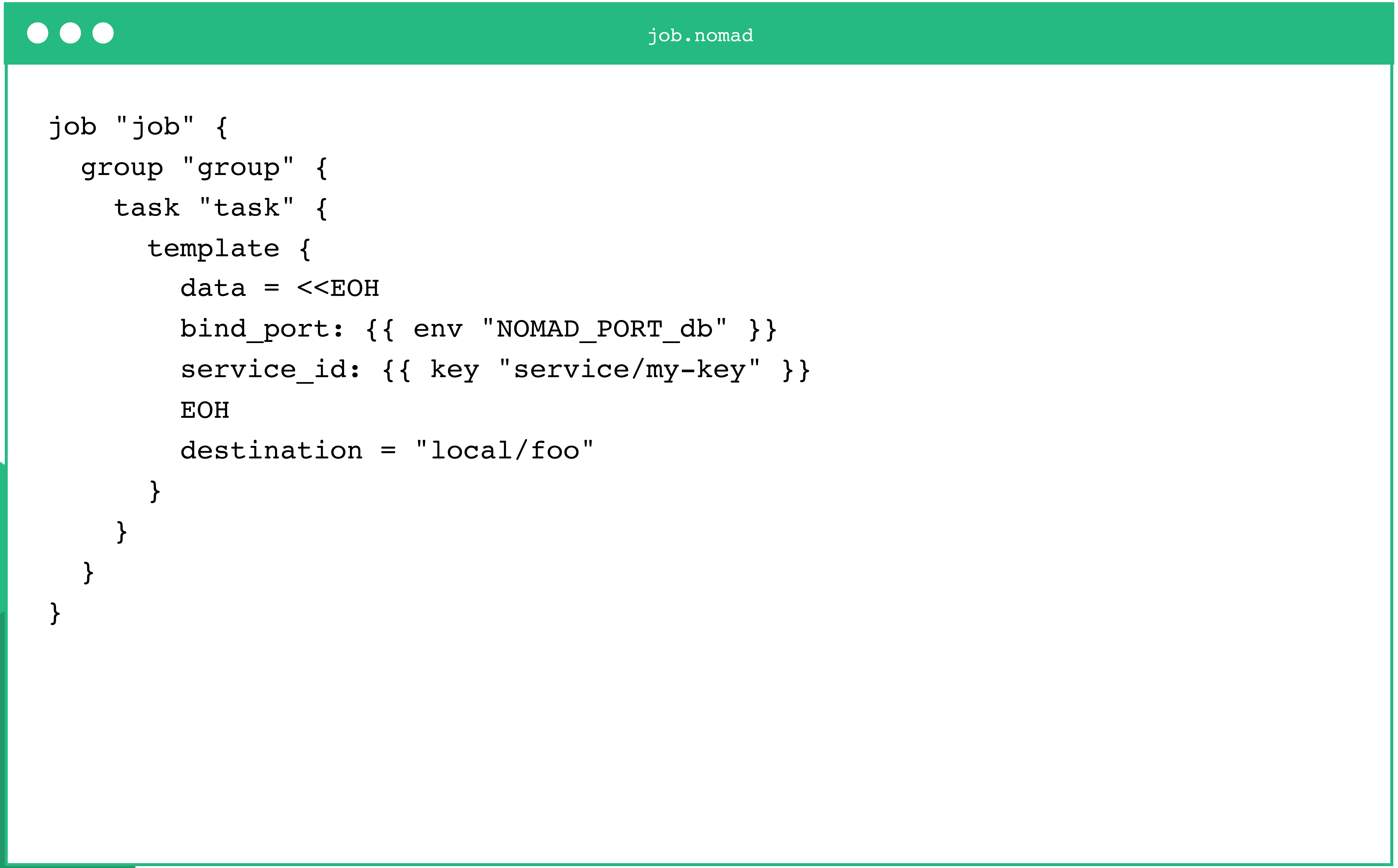
Templating

Nomad Templating

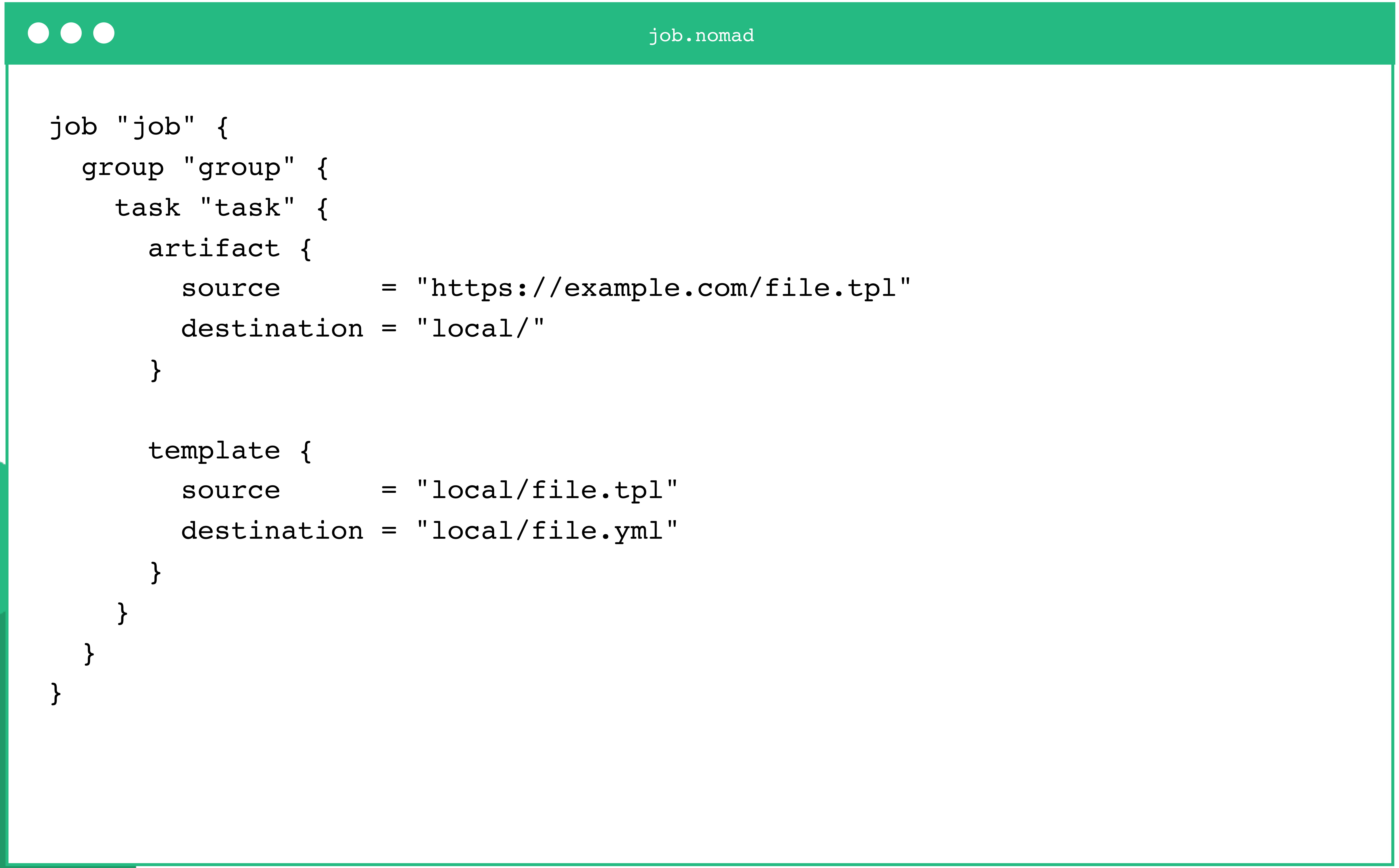
Nomad can take an input template, apply interpolations, and produce an output template for a task.

The template responds to signals and handles upstream updates.

Can query data from Nomad, Consul, and Vault.



```
job "job" {
  group "group" {
    task "task" {
      template {
        data = <<EOH
        bind_port: {{ env "NOMAD_PORT_db" }}
        service_id: {{ key "service/my-key" }}
        EOH
        destination = "local/foo"
      }
    }
  }
}
```



```
job "job" {
  group "group" {
    task "task" {
      artifact {
        source      = "https://example.com/file.tpl"
        destination = "local/"
      }

      template {
        source      = "local/file.tpl"
        destination = "local/file.yml"
      }
    }
  }
}
```

Advanced Job Strategies

Advanced Job Strategies

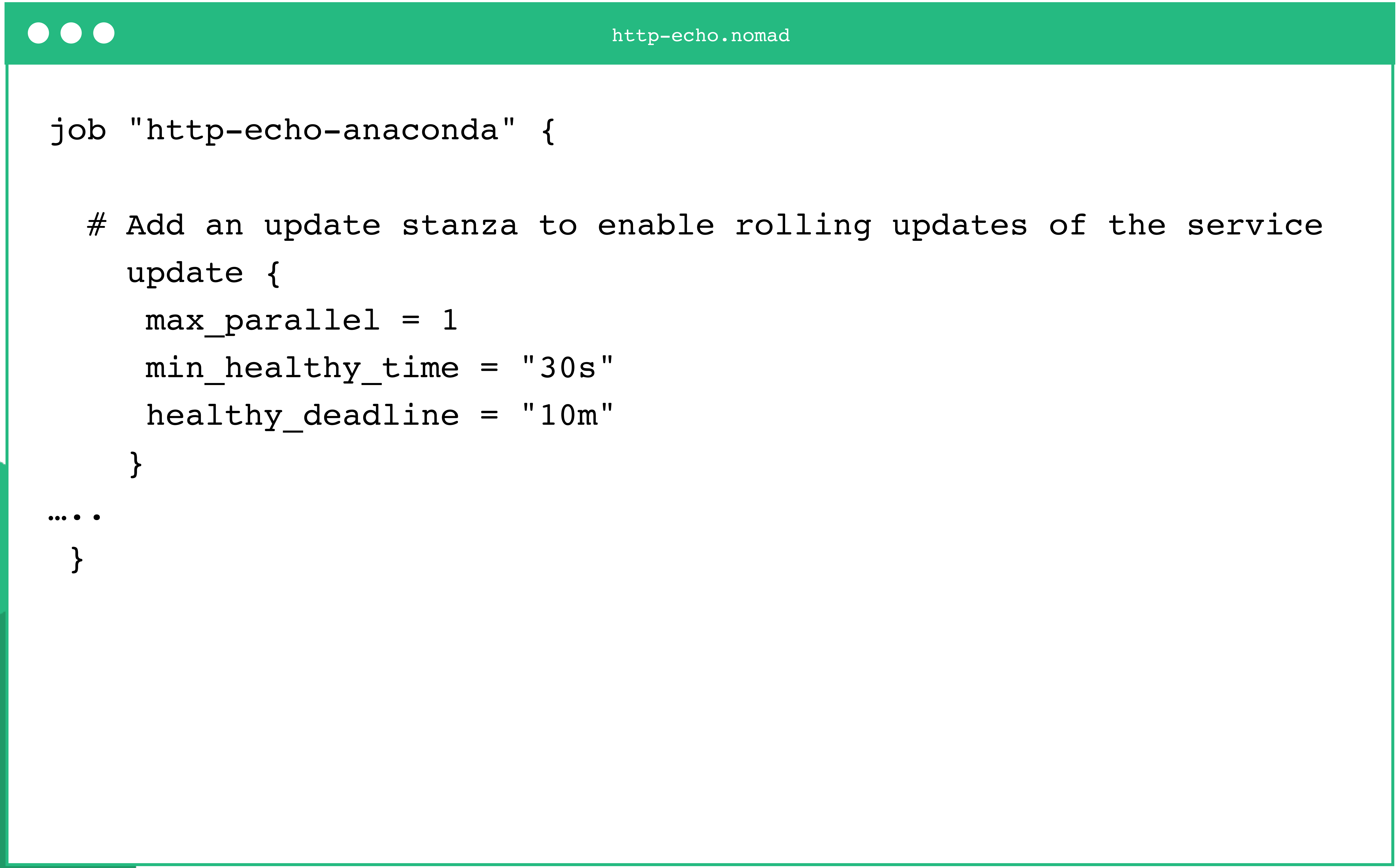
Nomad currently supports two advanced job strategies:

- Rolling Upgrades
- Blue/Green & Canary Deployments

Advanced Job Strategies: Rolling Upgrades

Nomad supports rolling updates as a first class feature.

To enable rolling updates a job or task group is annotated with a high-level description of the update strategy using the **`update`** stanza.



```
job "http-echo-anaconda" {

  # Add an update stanza to enable rolling updates of the service
    update {
     max_parallel = 1
     min_healthy_time = "30s"
     healthy_deadline = "10m"
    }
…..
 }
```

Exercise: Add update stanza to http-echo Job

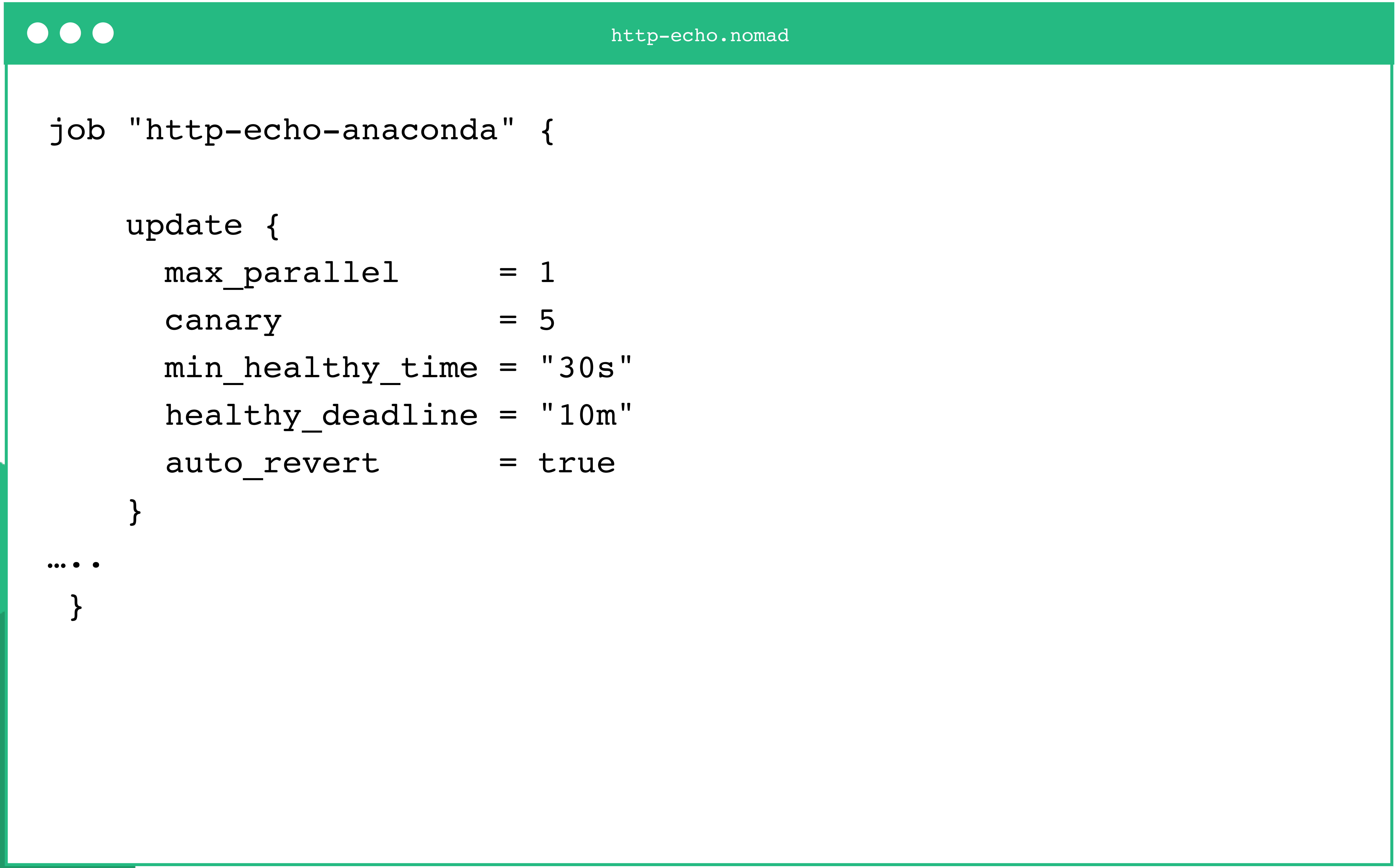
```
job "http-echo-anaconda" {

  # Add an update stanza to enable rolling updates of the service
    update {
     max_parallel = X
     min_healthy_time = "Xs"
    }
.....
 }
```

Open **http://nomad.hashicorp.live/** in your browser.

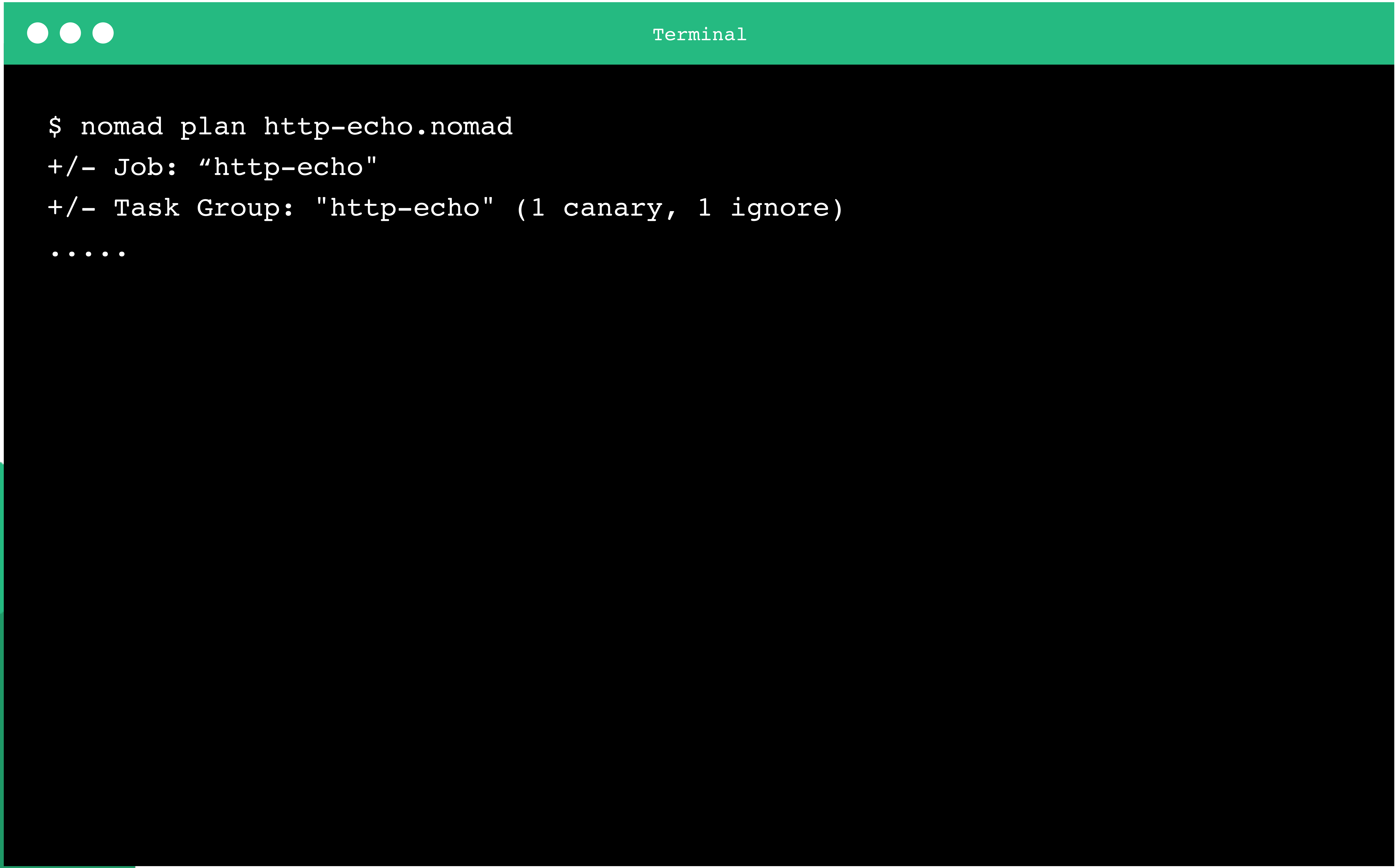
Advanced Job Strategies: Blue/Green & Canary

Sometimes rolling upgrades do not offer the required flexibility for updating an application in production.

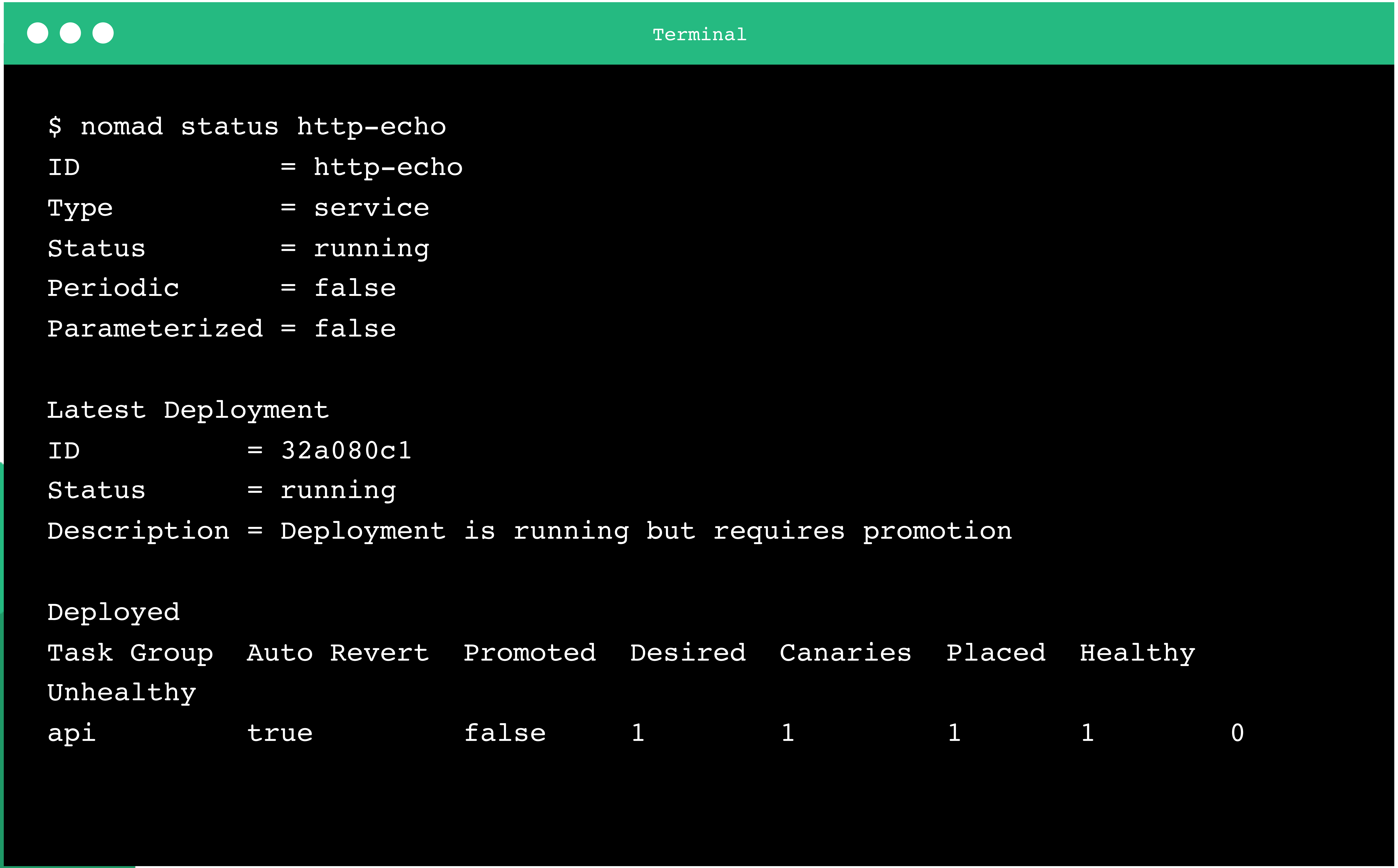
Often organizations prefer to put a "canary" build into production or utilize a technique known as a "blue/green" deployment to ensure a safe application rollout to production while minimizing downtime.



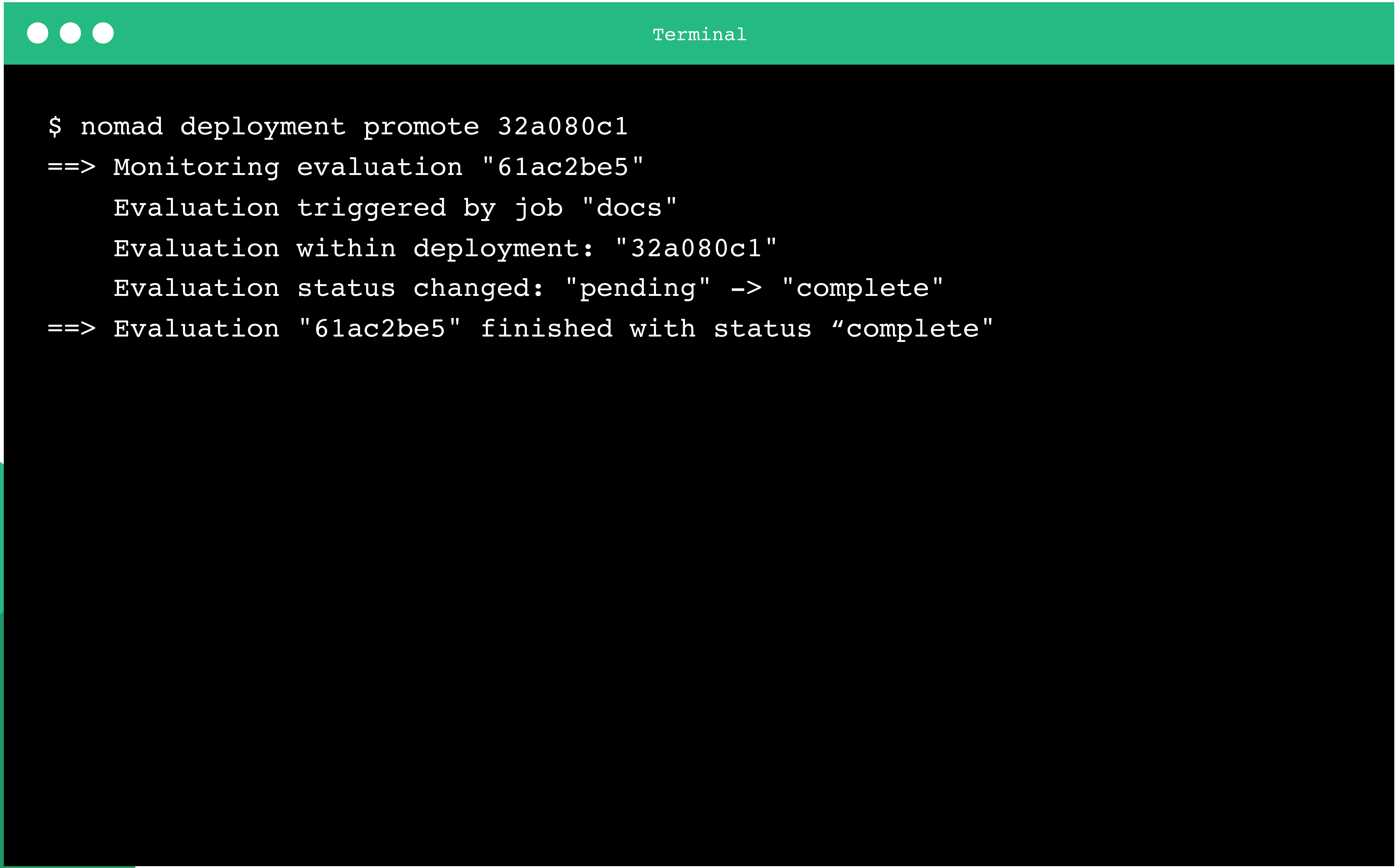
```
job "http-echo-anaconda" {

    update {
      max_parallel     = 1
      canary           = 5
      min_healthy_time = "30s"
      healthy_deadline = "10m"
      auto_revert      = true
    }
…..
 }
```



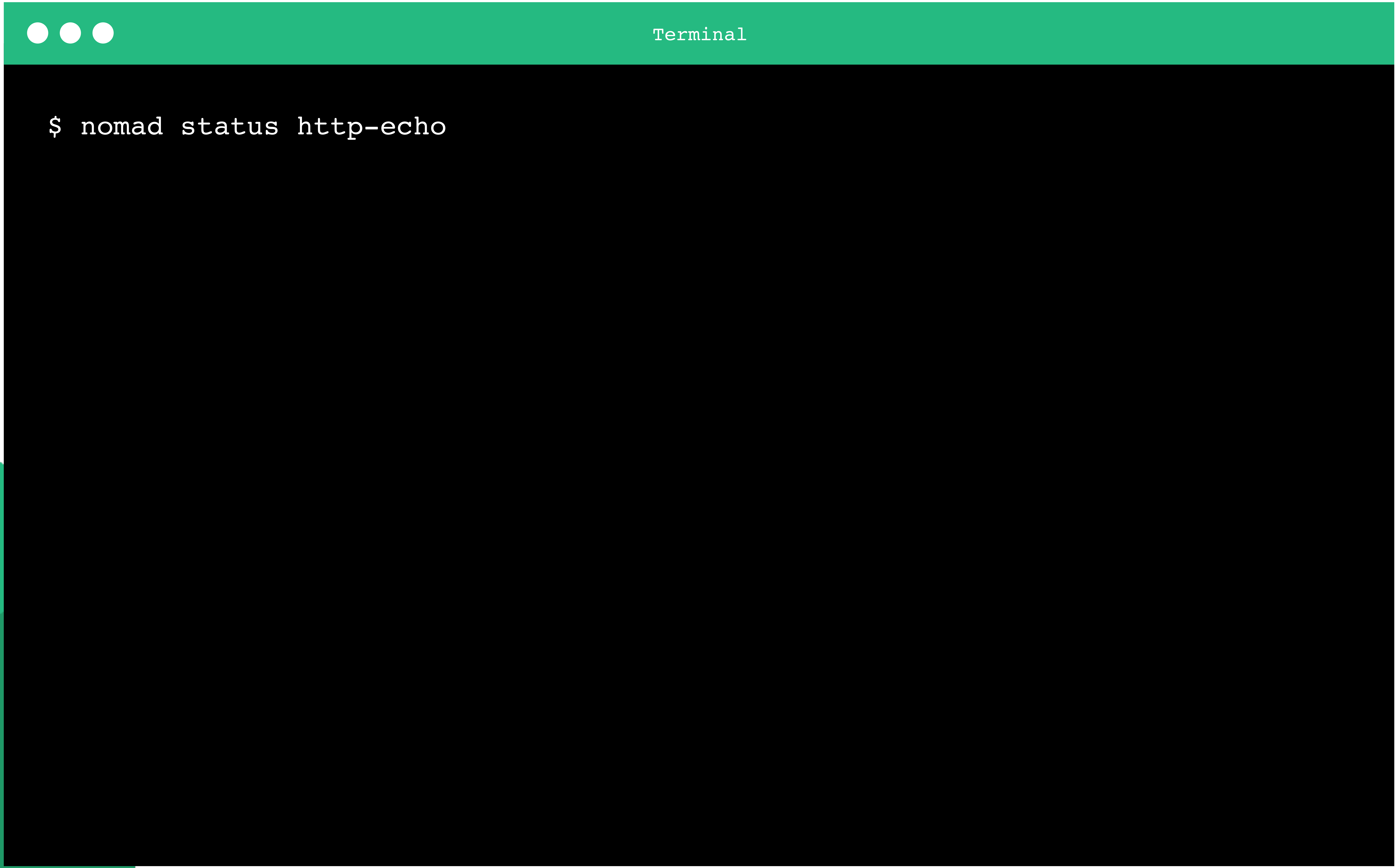
```
$ nomad plan http-echo.nomad
+/- Job: “http-echo"
+/- Task Group: "http-echo" (1 canary, 1 ignore)
.....
```



```
$ nomad status http-echo
ID            = http-echo
Type          = service
Status        = running
Periodic      = false
Parameterized = false

Latest Deployment
ID          = 32a080c1
Status      = running
Description = Deployment is running but requires promotion

Deployed
Task Group  Auto Revert  Promoted  Desired  Canaries  Placed  Healthy
Unhealthy
api         true         false     1        1         1       1        0
```



```
$ nomad deployment promote 32a080c1
==> Monitoring evaluation "61ac2be5"
    Evaluation triggered by job "docs"
    Evaluation within deployment: "32a080c1"
    Evaluation status changed: "pending" -> "complete"
==> Evaluation "61ac2be5" finished with status “complete"
```



```
$ nomad status http-echo
```

Nomad UI

http://nomad.hashicorp.live/

Demo

Q&A