Nick Ethier

Nomad & Lessons Learned Building Large Distributed Systems in Go

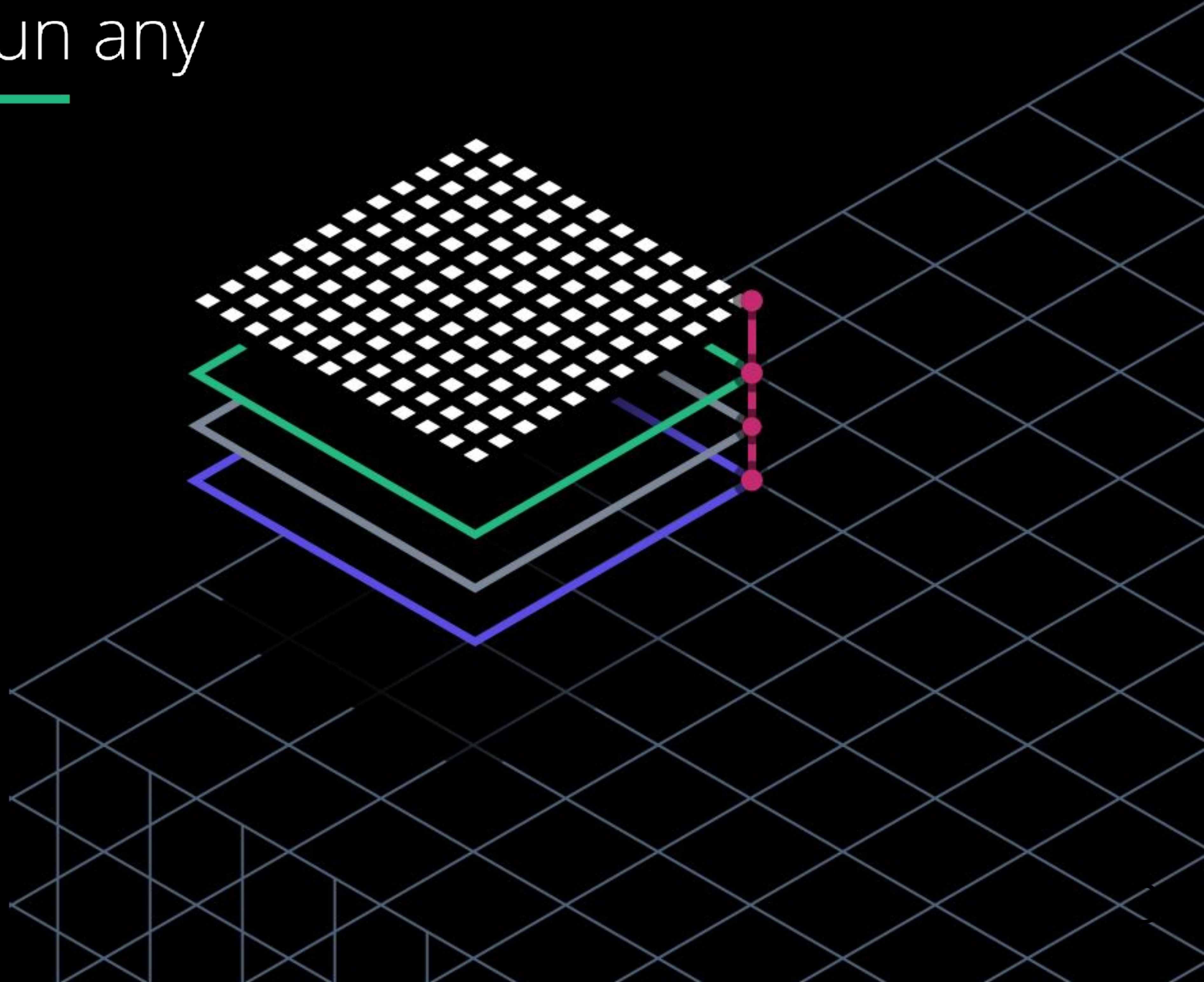
Agenda

- HashiCorp Intro (5 min)
- What is Nomad? (15 min)
- Lessons Learned (20 min)

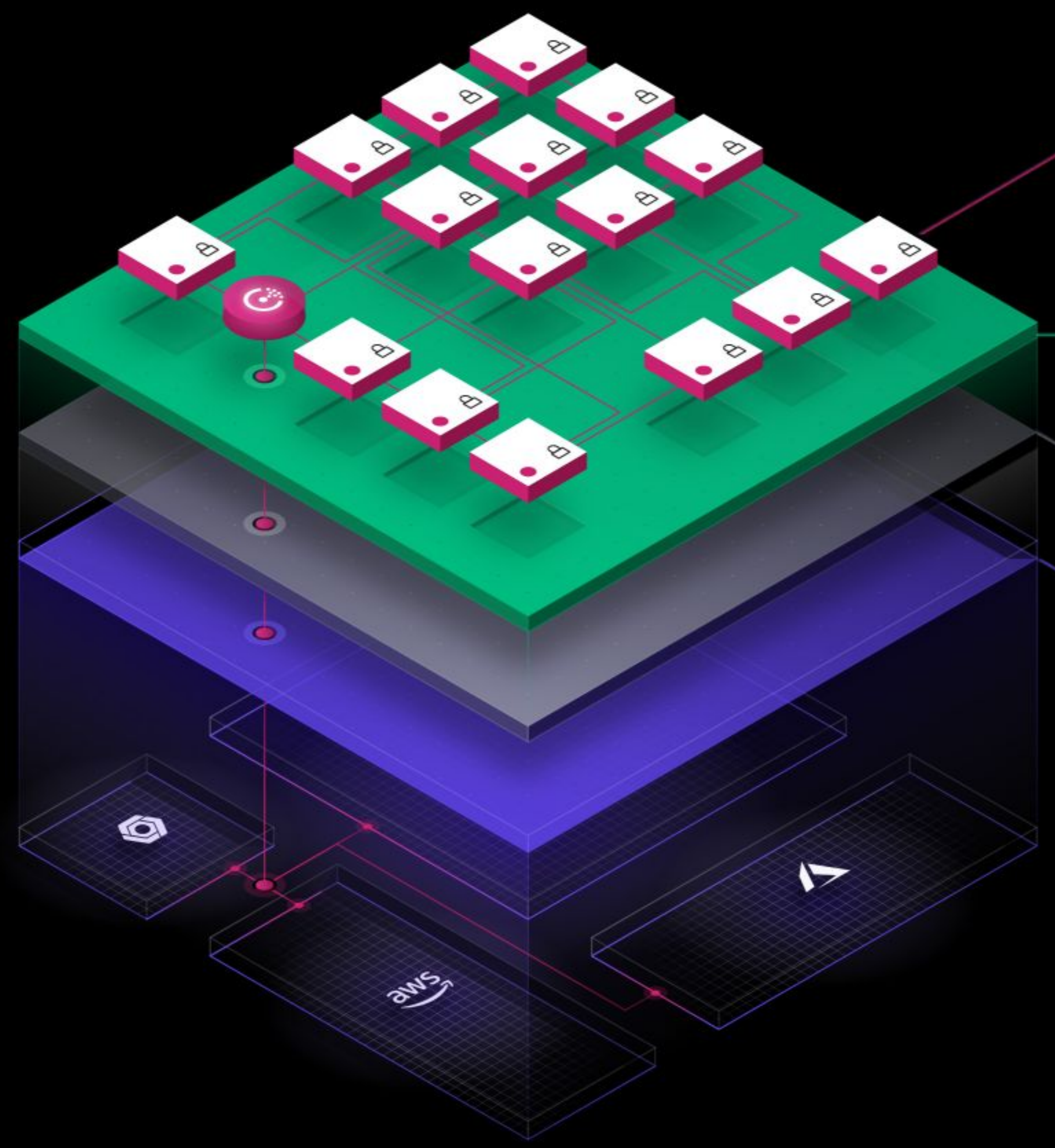
HashiCorp

Cloud Infrastructure Automation

Consistent workflows to provision, secure, connect, and run any infrastructure for any application.



The 4 essential elements of dynamic infrastructure



Networking

Connect infrastructure and applications

Development

Run applications

Security

Secure applications and infrastructure

Operations

Provision Infrastructure

Provision

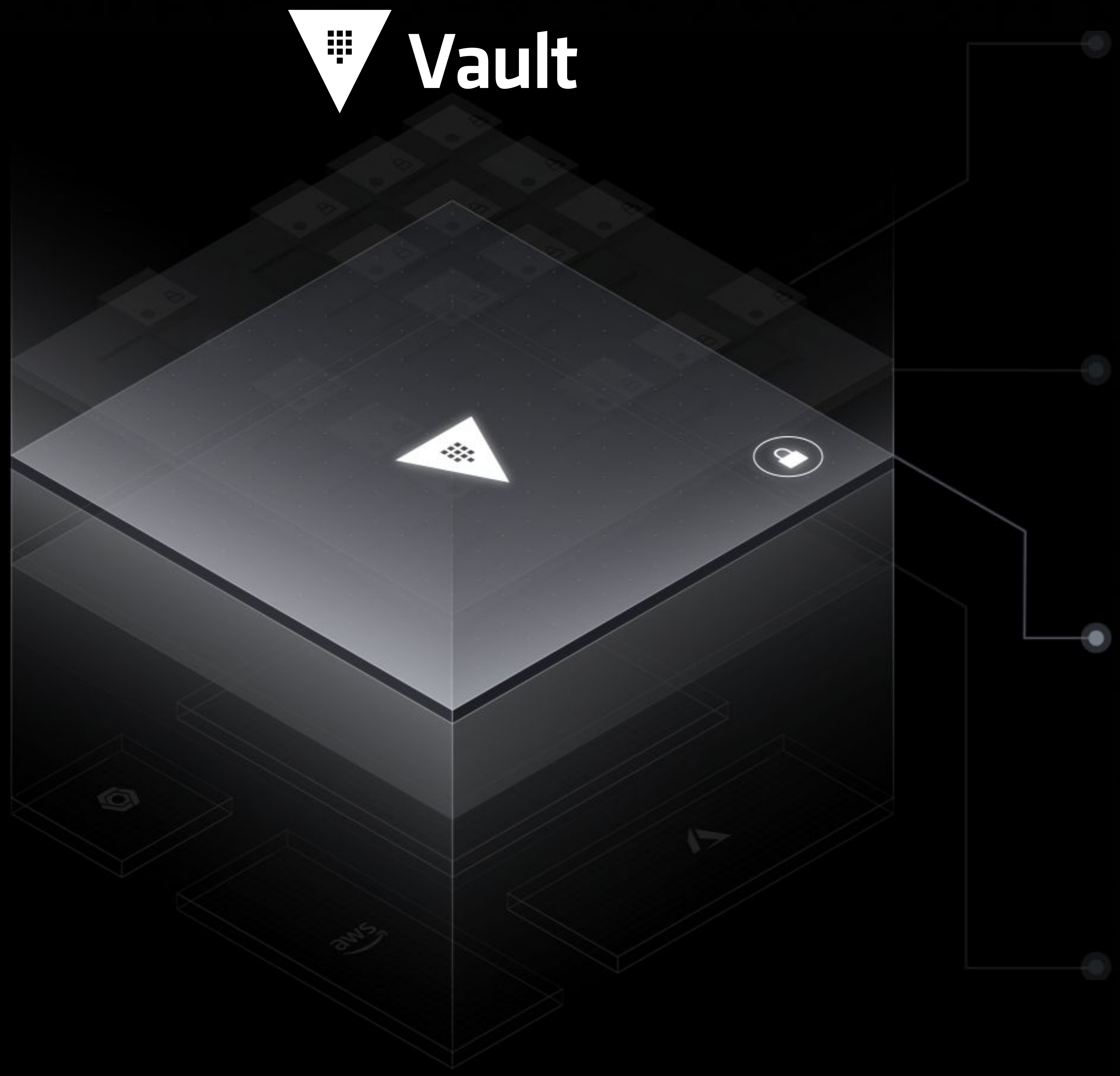
Write, Plan and Create Infrastructure as Code

Operations

Provision Infrastructure

Secure

Store and Control Access to Secrets

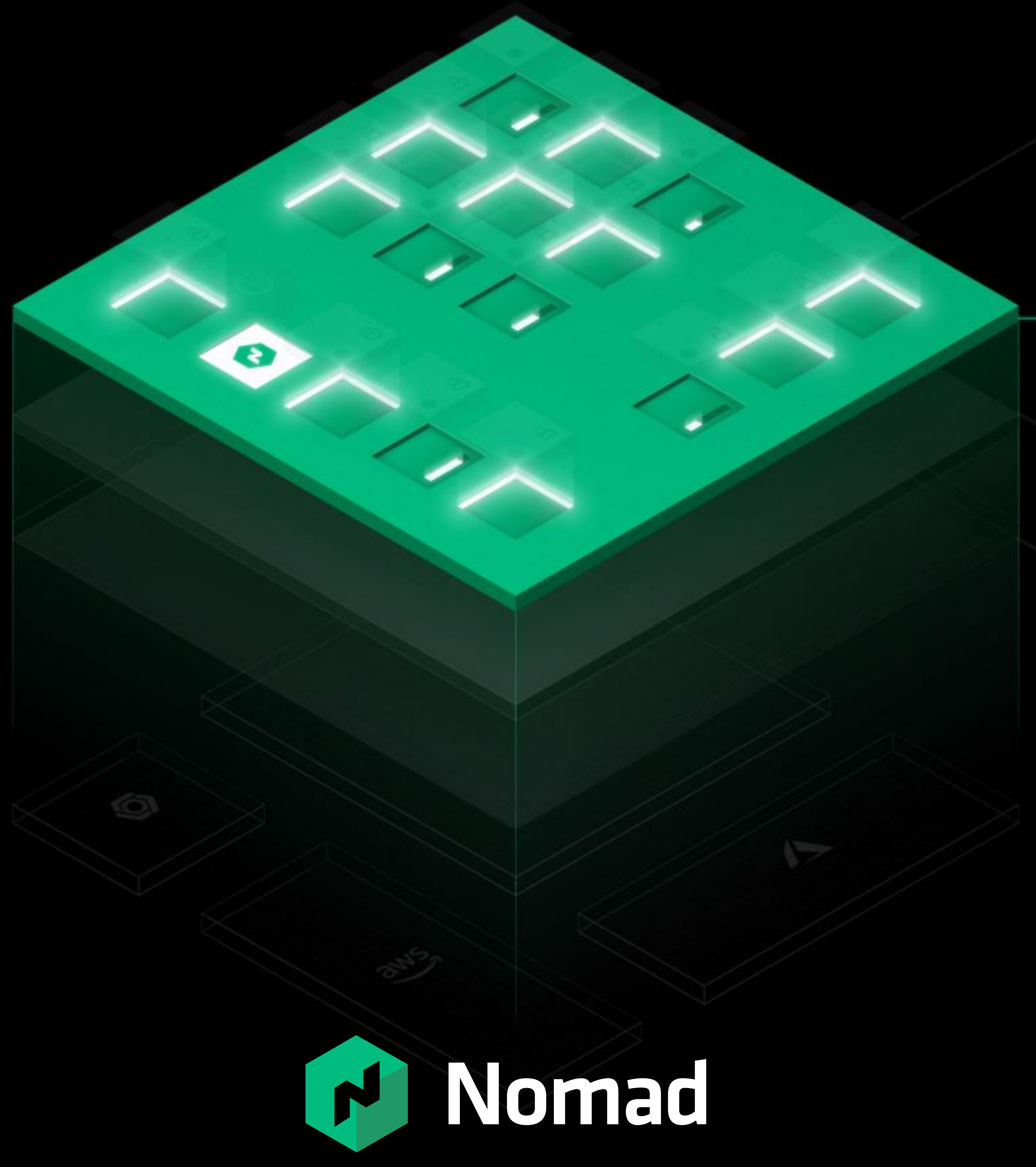


Security

Secure applications and infrastructure

Run

Easily Deploy Applications at Any Scale

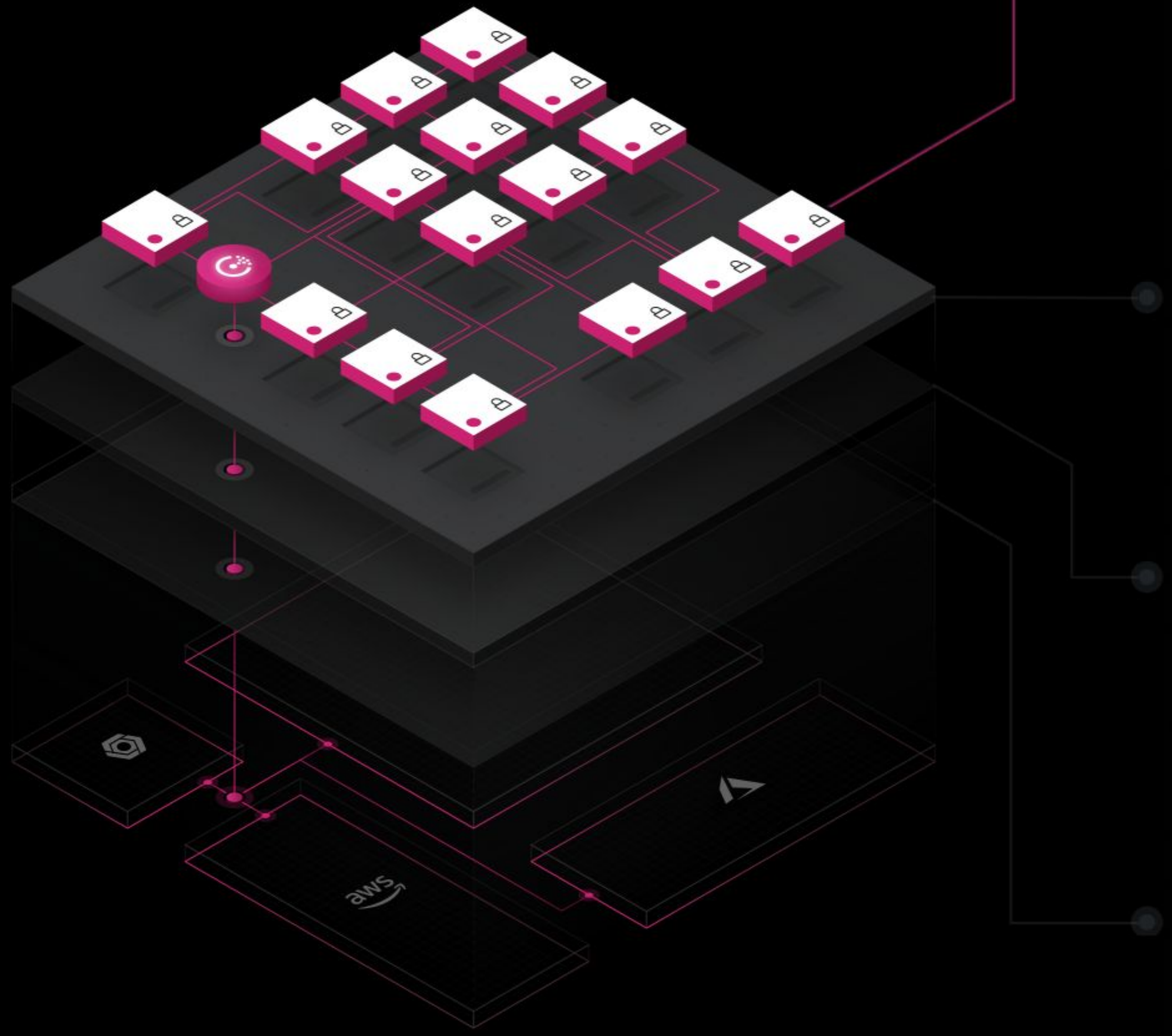


Development

Run Applications

Connect

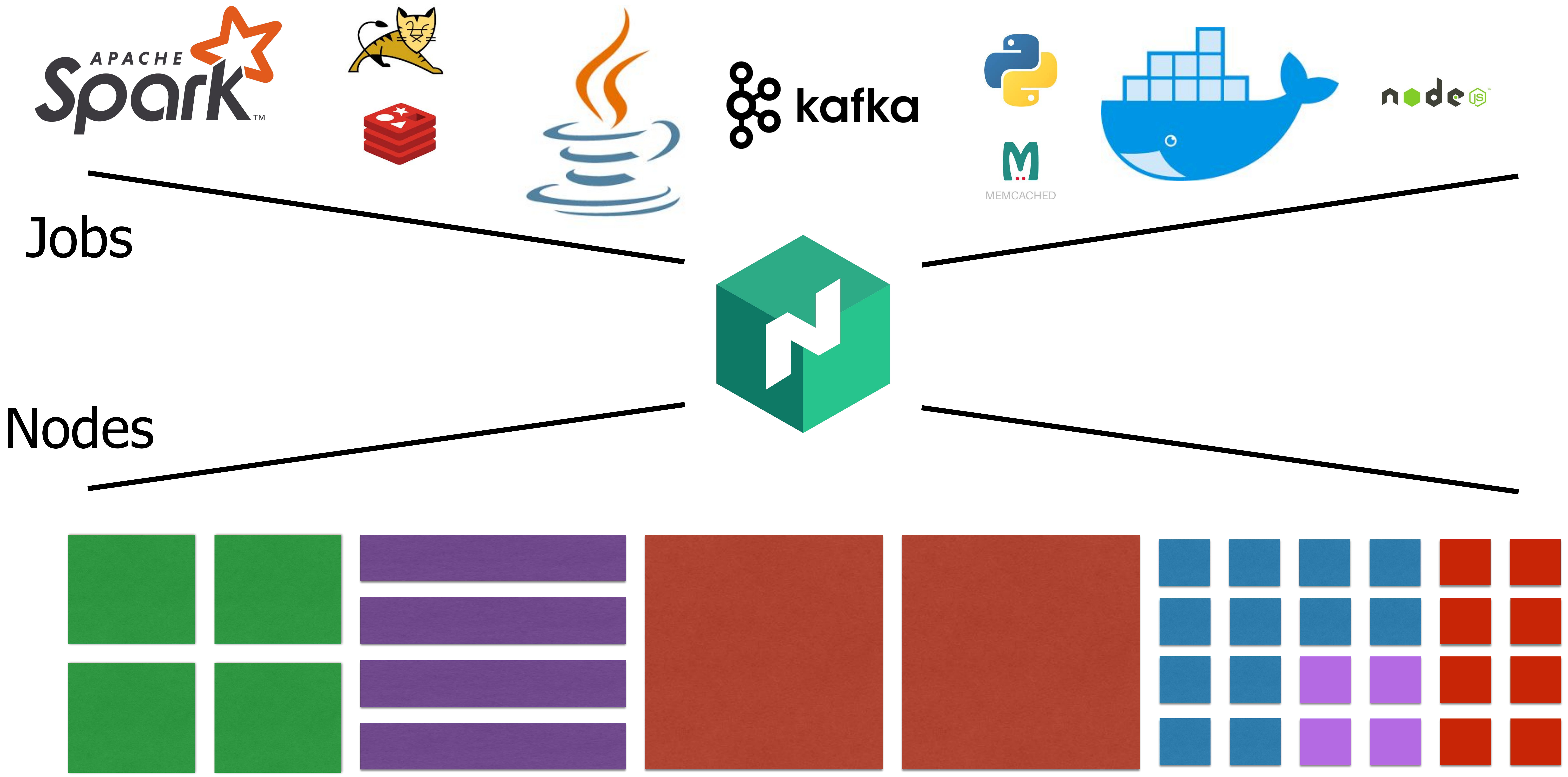
Service Mesh Made
Easy



Consul

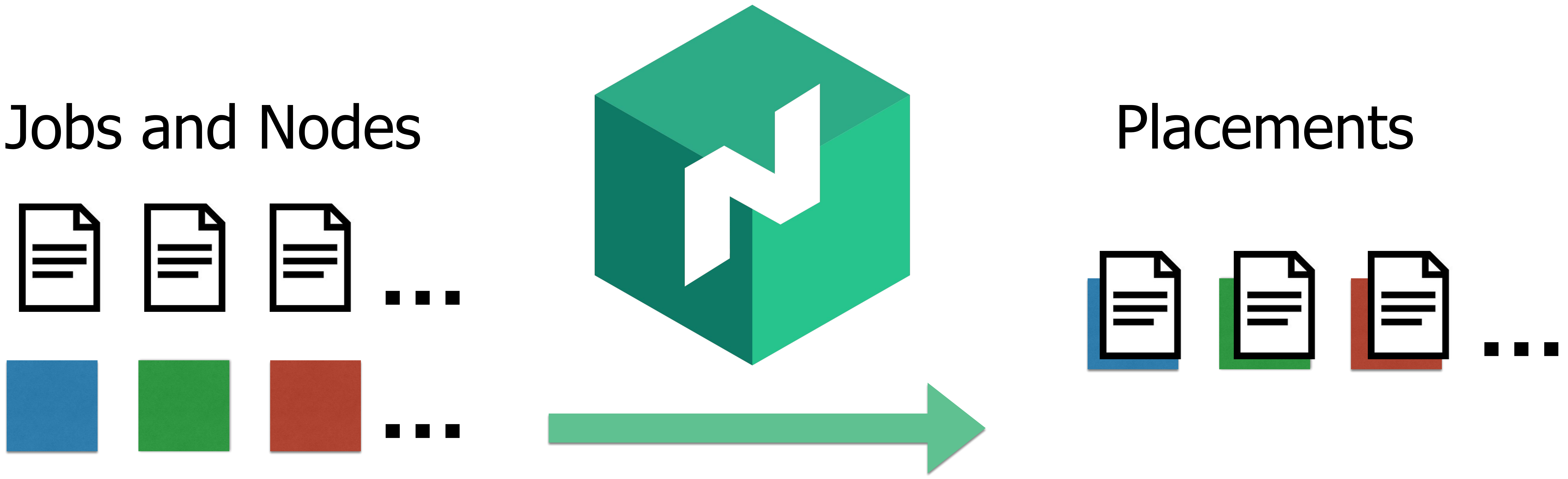
Networking

Connect Infrastructure and
Applications



HashiCorp

Scheduler: Map Work to Resources



Schedulers

- Service
  - Long running tasks
  - Tasks should be restarted if die
- Batch
  - Short term tasks
  - Tasks typically run until completion
  - Supports cron style scheduling
- System
  - Much like service scheduler
  - One placement for every client node

Job File

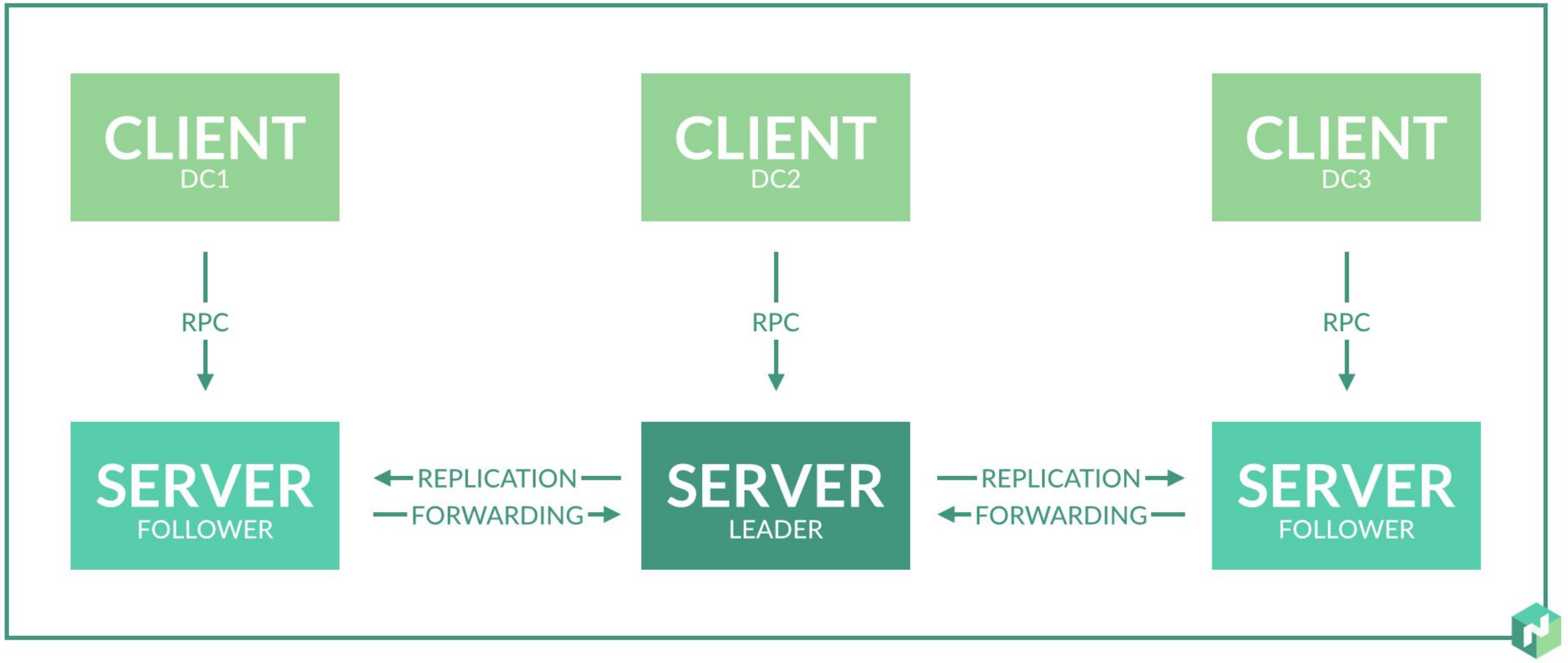
redis.nomad

```
job "cache" {
  datacenters = ["dc1"]
  group "cache" {
    task "redis" {
      driver = "docker"

      config {
        image = "redis:3.2"
        port_map {
          db = 6379
        }
      }

      resources {
        cpu    = 500
        memory = 256
        network {
          port "db" {}
        }
      }
    }
  }
}
```

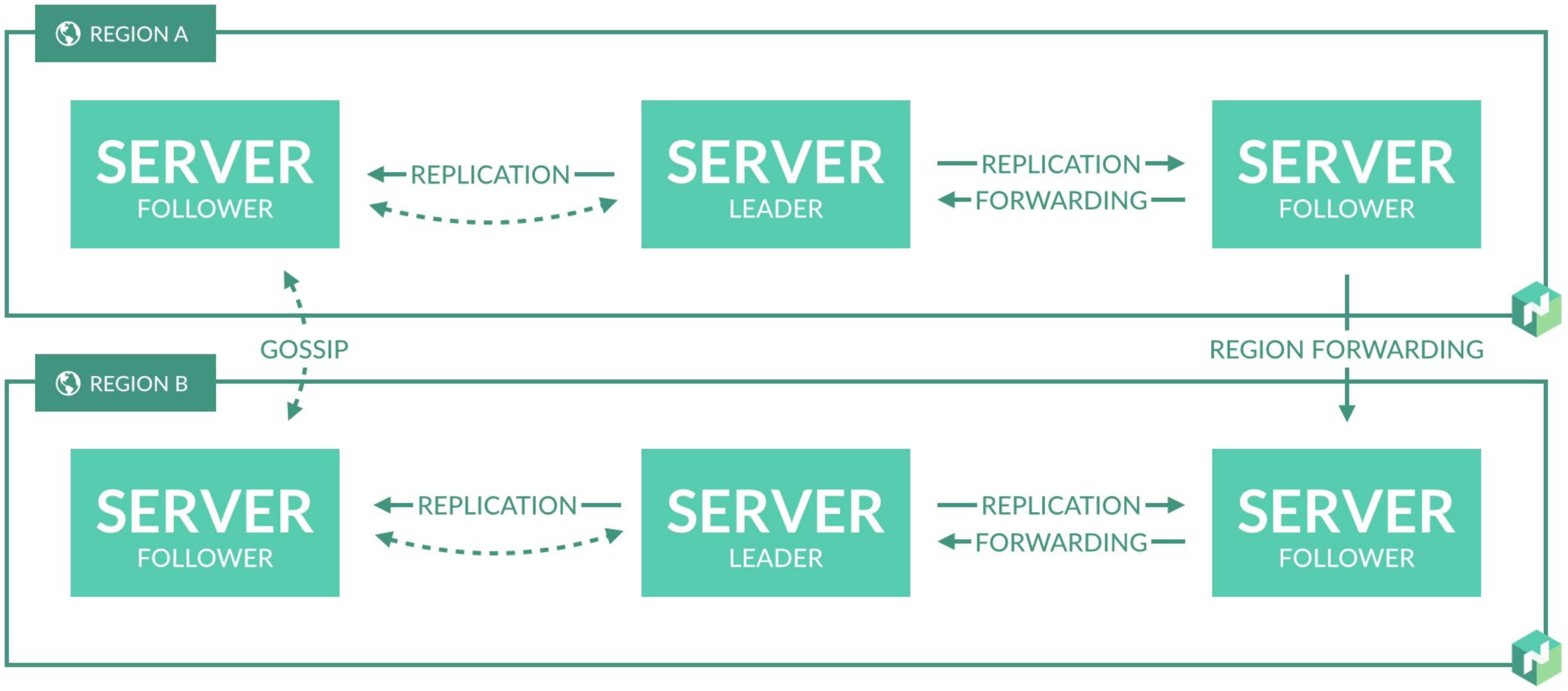
Nomad Core Tenants

- Workflows not Technologies
  - Containerized and Legacy Applications
  - Pluggable Drivers
- Easy to Use
  - No external state store
  - Single Binary
- Scalable and Performant
  - 10,000+ Node Deployments
  - Schedule 5,500 Placements a Second (1M/18.1s)

Architecture



Multi-Region



Public Users

SAP Ariba

deluxe®

PANDORA®

CITADEL

jet

circleci

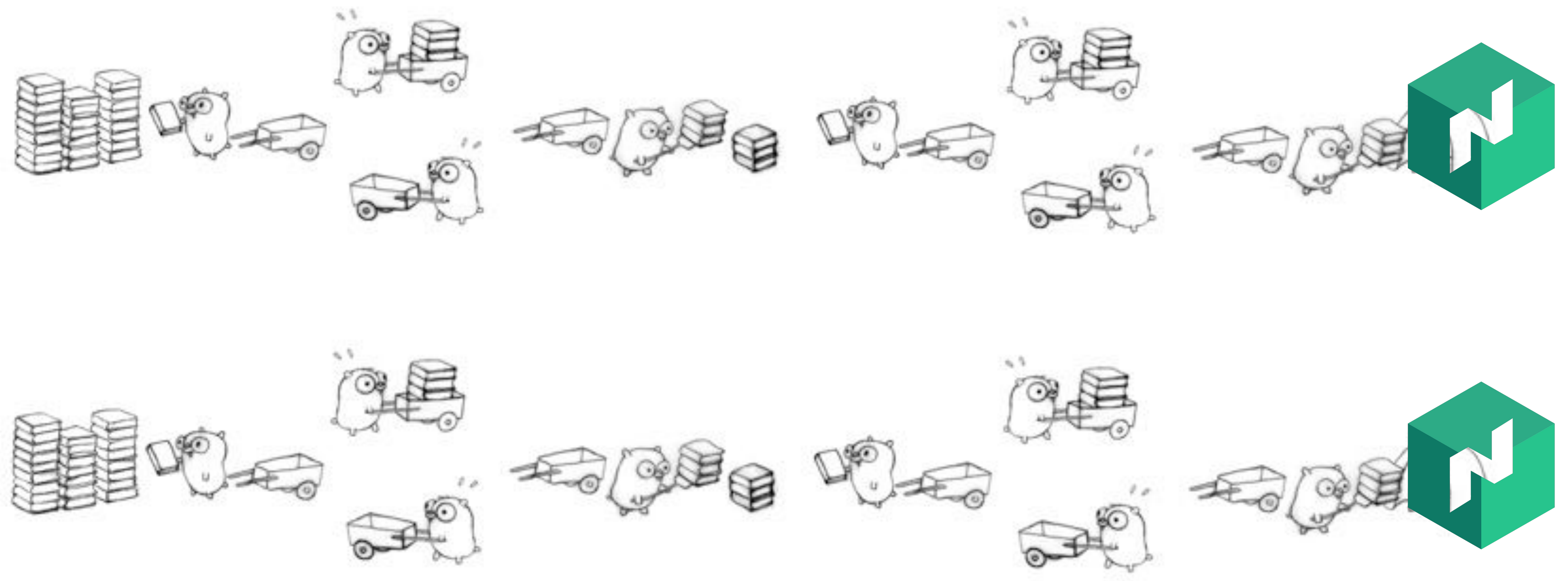
SPACEFLIGHT

Demo Time

HashiCorp

Nomad and Golang

- Many thousands/tens of thousands of goroutines
- Coordinated startup/shutdown of many concurrent components
- Retry all the things!
- Build and test all the platforms



Goroutines

- Like a thread but very cheap
- Executes in the same address space as caller
- Can leak!!
- Mapped to OS threads but don’t block them

```
foo() // blocks until foo completes

go foo()
bar() // foo starts but doesn’t block
```

Gotchas: Gorountine

```
func main() {
  books := []string{
    "The Lord of the Rings: The Two Towers",
    "Harry Potter and the Order of the Phoenix",
    "Twilight",
  }
  for _, book := range books {
    go func() {
      openBook(book)
      readBook(book)
    }()
  }
}

func openBook(book string) {
  fmt.Println("Opening: " + book)
}
func readBook(book string) {
  fmt.Println("Reading: " + book)
}
```

Gotchas: Gorountine

```
func TestMyFunc(t *testing.T){
  cases := []struct{
    input string
    output string
  }{
    {“foo”, “bar”},
    {“abc”,”xyz”},
  }

  for i, c := range cases {
    t.Run(fmt.Sprintf(“Case-%d”, i), func(t *testing.T){
      if c.output != MyFunc(c.input) {
        t.Fail(“expected output did not match”)
      }
    })
  }
```

Channels

- Used to communicate between channels
- Sending and receiving on a channel blocks
- Can be buffered
- Can leak, make sure you close your channels!!!

```
ch := make(chan string)

go func(){
  ch <- “Hello from a goroutine”
  close(ch)
}()

fmt.Println(<-ch)
```

Select

- Like a switch statement for channels
- All cases are evaluated until one unblocks

```
ch1 := make(chan string)
ch2 := make(chan string)
go worker(ch1)
go worker(ch2)

select {
case s := <-ch1:
  fmt.Println("ch1: " + s)
case s := <-ch2:
  fmt.Println("ch2: " + 2)
}
```

Gotchas: Select

```
func worker(stopCh chan struct{}, queue chan *Work) {
  for {
    select {
    case w := <-queue:
      // Do work
    case <-stopCh:
      return
    }
  }
}
```

Gotchas: Select

```
func worker(stopCh chan struct{}, queue chan *Work) {
  for {
    select {
    case w, ok := <-queue:
      if !ok {
        return
      }
      // Do work
    case <-stopCh:
      return
    }
  }
}
```

Using a channel to signal

```go
func watchChanges(signalCh <-chan struct{}) {
  timer := time.NewTimer(0)
  for {
    select {
    case <-timer.C: {
      updateServer()
      timer.Reset(time.Second * 30)
    case <-signalCh:
      updateServer()
    }
  }
}
```

```go
func runWorker(shutdown <-chan struct{}) {
  for {
    select {
    case <-shutdown: {
      return
    case job <-nextJob():
      // do work
    }
  }
}
func main() {
  ch := make(chan struct{})
  defer close(ch)
  for i := 0; i<10; i++ {
    runWorker(ch)
  }
  time.Sleep(time.Second * 30)
}
```

Context

- More powerful signaling utility
- Carries deadline, cancelation and other request scoped values

```
func handleRequest(ctx context.Context, req *Request) error {
  doneCh := make(chan struct{})
  go func(){
    defer close(doneCh)
    doExpensiveFunc()
  }

  select {
  case <-time.After(time.Second * 30):
    return fmt.Errorf("timed out")
  case <-ctx.Done():
    return ctx.Err()
  case <-doneCh:
    return nil
}
```

Context

- More powerful signaling utility
- Carries deadline, cancelation and other request scoped values

```
func makeRequest() error {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()

  handleRequest(ctx, &Request{})
}

func makeRequest() error {
  ctx, cancel := context.WithTimeout(context.Background(),
    time.Second * 10)
  defer cancel()

  handleRequest(ctx, &Request{})
}
```

WaitGroup

- Waits for a collection of goroutines to fin
- Useful for parallelizing goroutines

```
func (s *Server) Start(){
  var wg sync.WaitGroup

  startFuncs := []func(){
    startHTTP
    startFingerprinter
    startPluginManager
  }

  for _, f := range startFuncs {
    wg.Add(1)
    fn := f
    go func(){
      defer wg.Done()
      fn()
    }
  }
  wg.Wait()
  log.Info("start up complete")
}
```

Thank you

hello@hashicorp.com

www.hashicorp.com

WHEN SOMEONE MENTIONS
GOLANG PACKAGE MANAGEMENT
memegenerator.net