Efficient Integration Testing in Go

A Case Study on Dapr

Josh van Leeuwen

Agenda

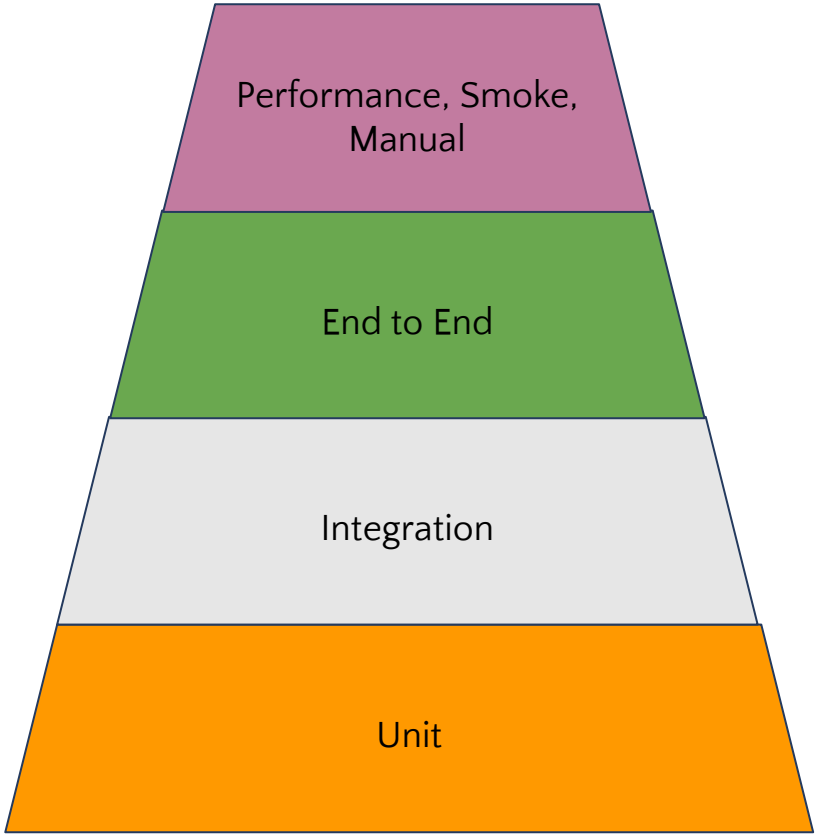
- TESTING
- DAPR
- FRAMEWORK
- NAMING (hard)
- PROCESS
  - (wrap)
  - bin
  - p|pe
- Assert eventually
- CLEANUP (really)
- OS
- Being Productive

TESTING - aka why are we here?, 42 etc.

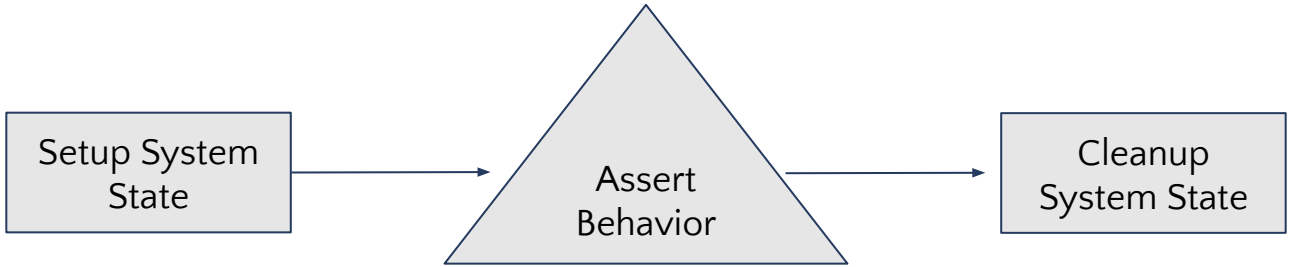
Why do we test software?

1. Prove correctness of software
2. Guardrails when changing implementation code
3. Ensure compatibility with external changing modules/APIs
4. Verify performance
5. Provide a framework for finding bugs and experimenting with features
6. **Increase velocity of development**

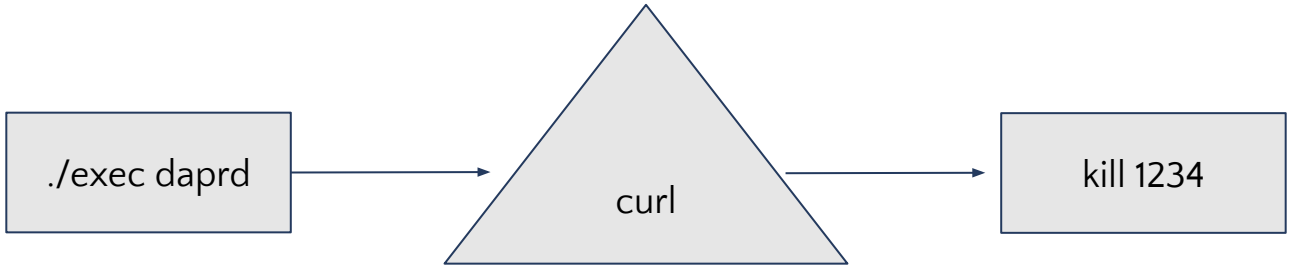
Test Types



Integration Testing

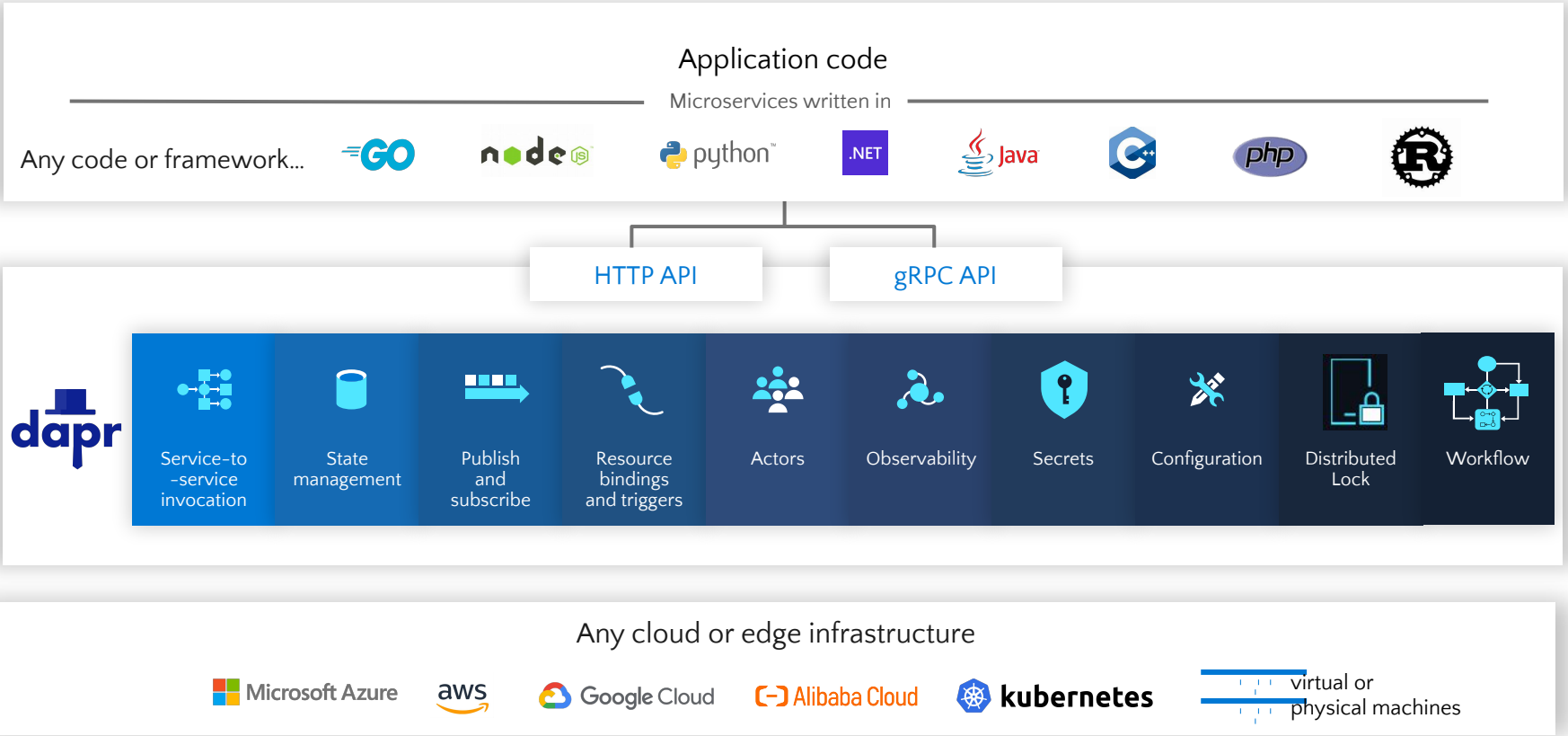


Integration Testing

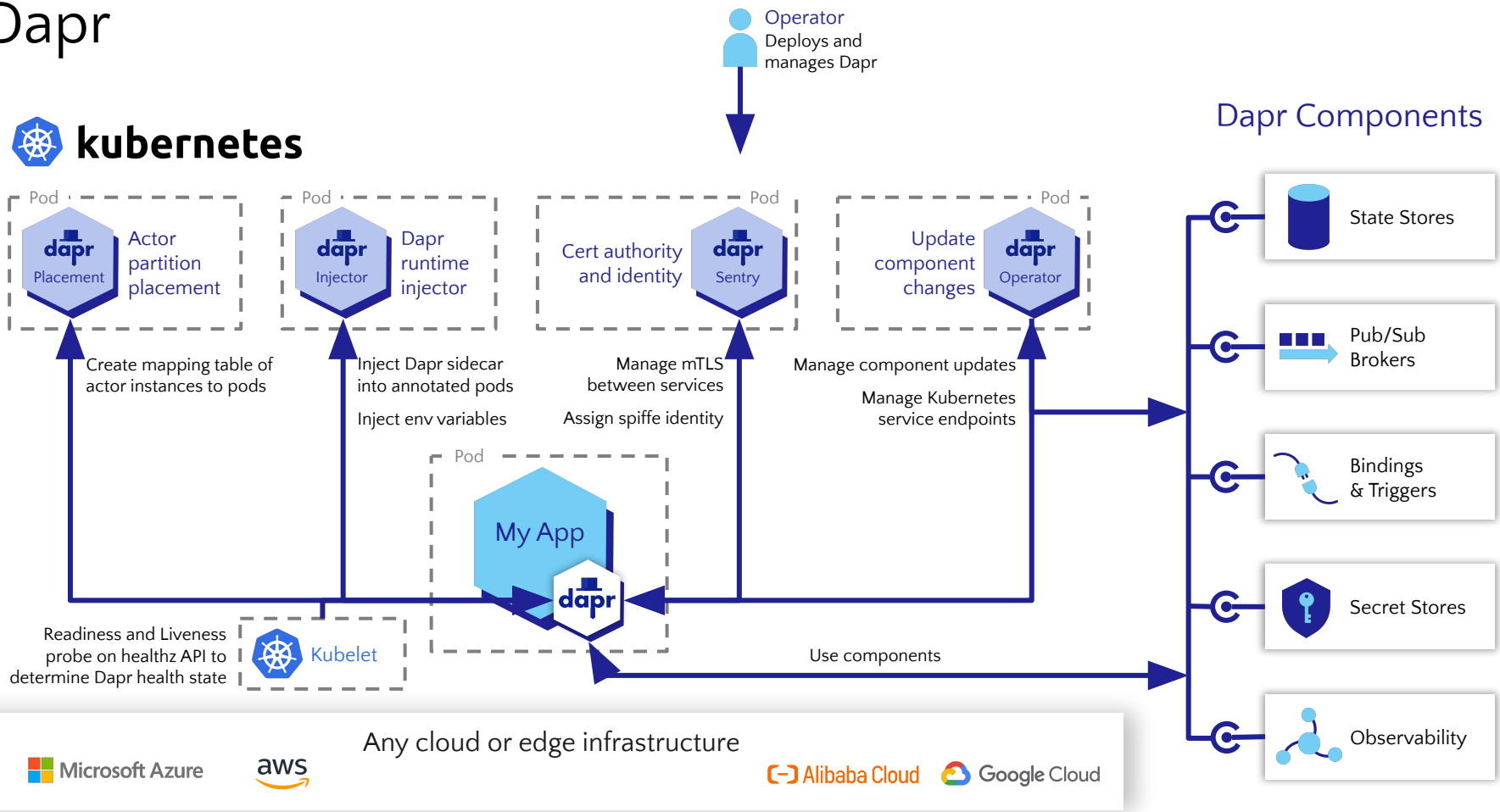


DAPR

Dapr APIs


Application code
Microservices written in
Any code or framework...
HTTP API
gRPC API
dapr
Service-to-service invocation
State management
Publish and subscribe
Resource bindings and triggers
Actors
Observability
Secrets
Configuration
Distributed Lock
Workflow
Any cloud or edge infrastructure
Microsoft Azure
aws
Google Cloud
Alibaba Cloud
kubernetes
virtual or physical machines

Dapr

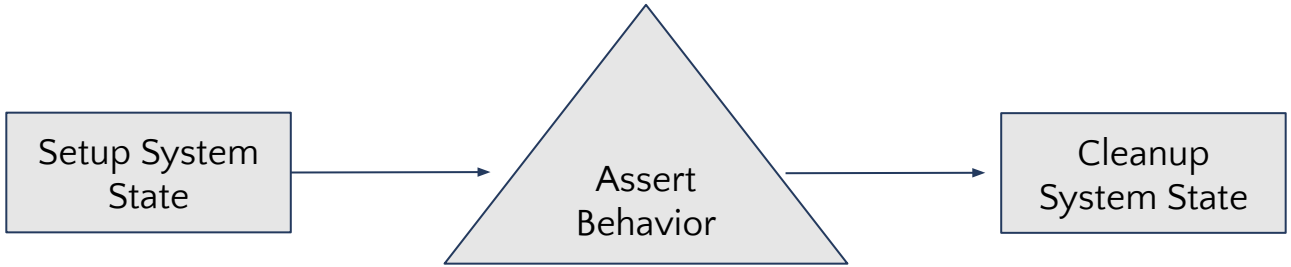

Operator
Deploys and manages Dapr
kubernetes
Dapr Components
Pod
dapr Placement
Actor partition placement
Pod
dapr Injector
Dapr runtime injector
Pod
Cert authority and identity
dapr Sentry
Pod
Update component changes
dapr Operator
Create mapping table of actor instances to pods
Inject Dapr sidecar into annotated pods
Inject env variables
Manage mTLS between services
Assign spiffe identity
Manage component updates
Manage Kubernetes service endpoints
Pod
My App
dapr
Readiness and Liveness probe on healthz API to determine Dapr health state
Kubelet
Use components
State Stores
Pub/Sub Brokers
Bindings & Triggers
Secret Stores
Observability
Any cloud or edge infrastructure
Microsoft Azure
aws
Alibaba Cloud
Google Cloud

Dapr Integration Design Decisions

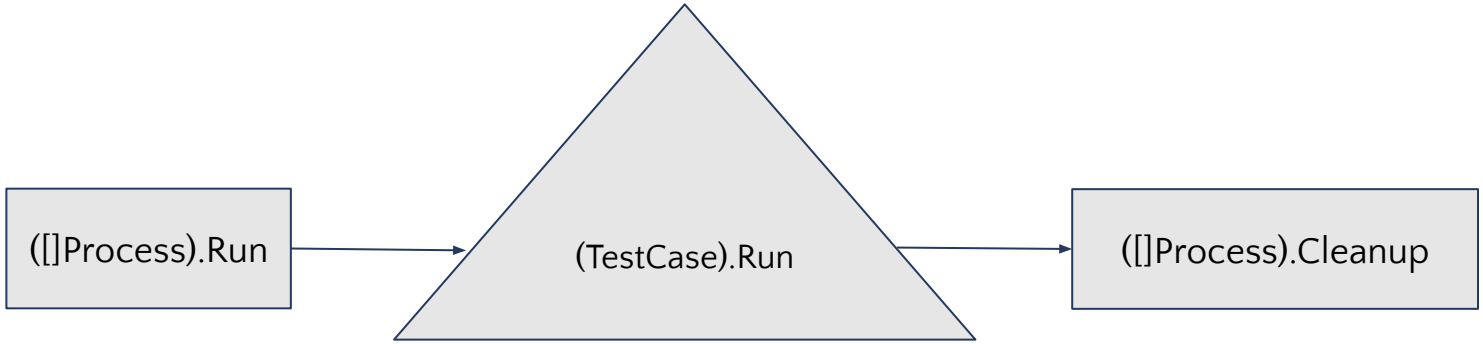
1. Go as sole dependency
2. “Quick” to execute (time.Sleep is banned..ish)
3. Portable
4. Extensible
5. Readable

FRAMEWORK

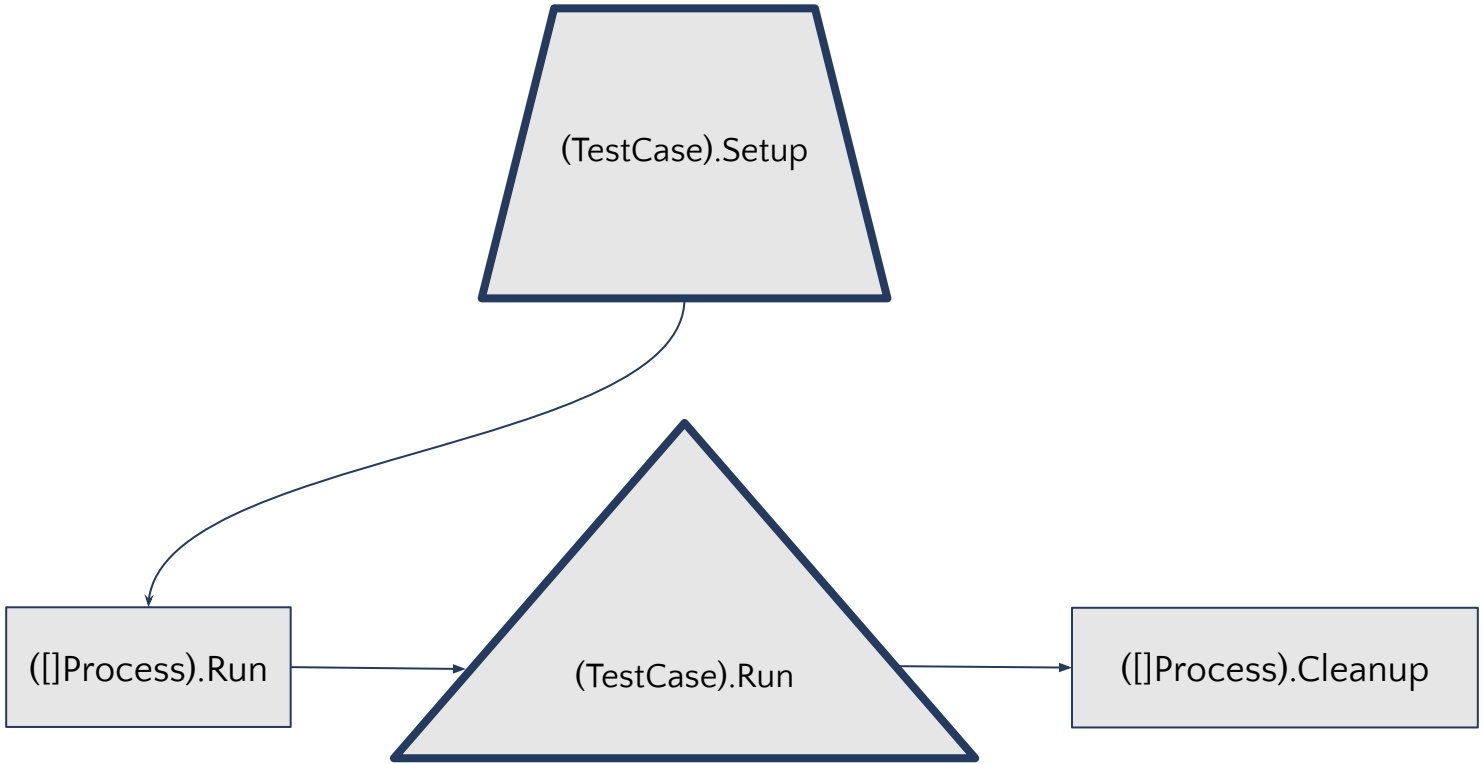
Integration Testing



Integration Testing



Test Case

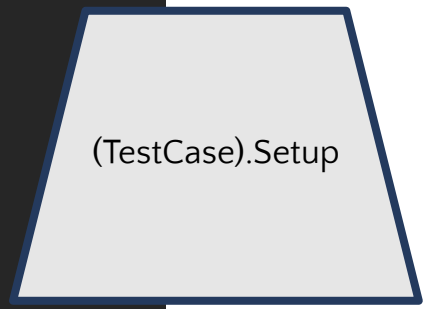


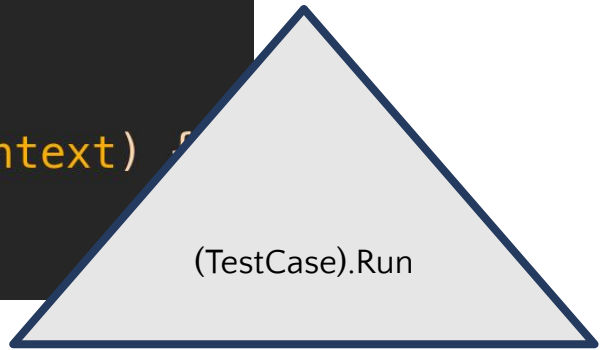
Test Case

```
// Case is a test case for the integration test suite.
type Case interface {
  Setup(*testing.T) []framework.Option
  Run(*testing.T, context.Context)
}
```

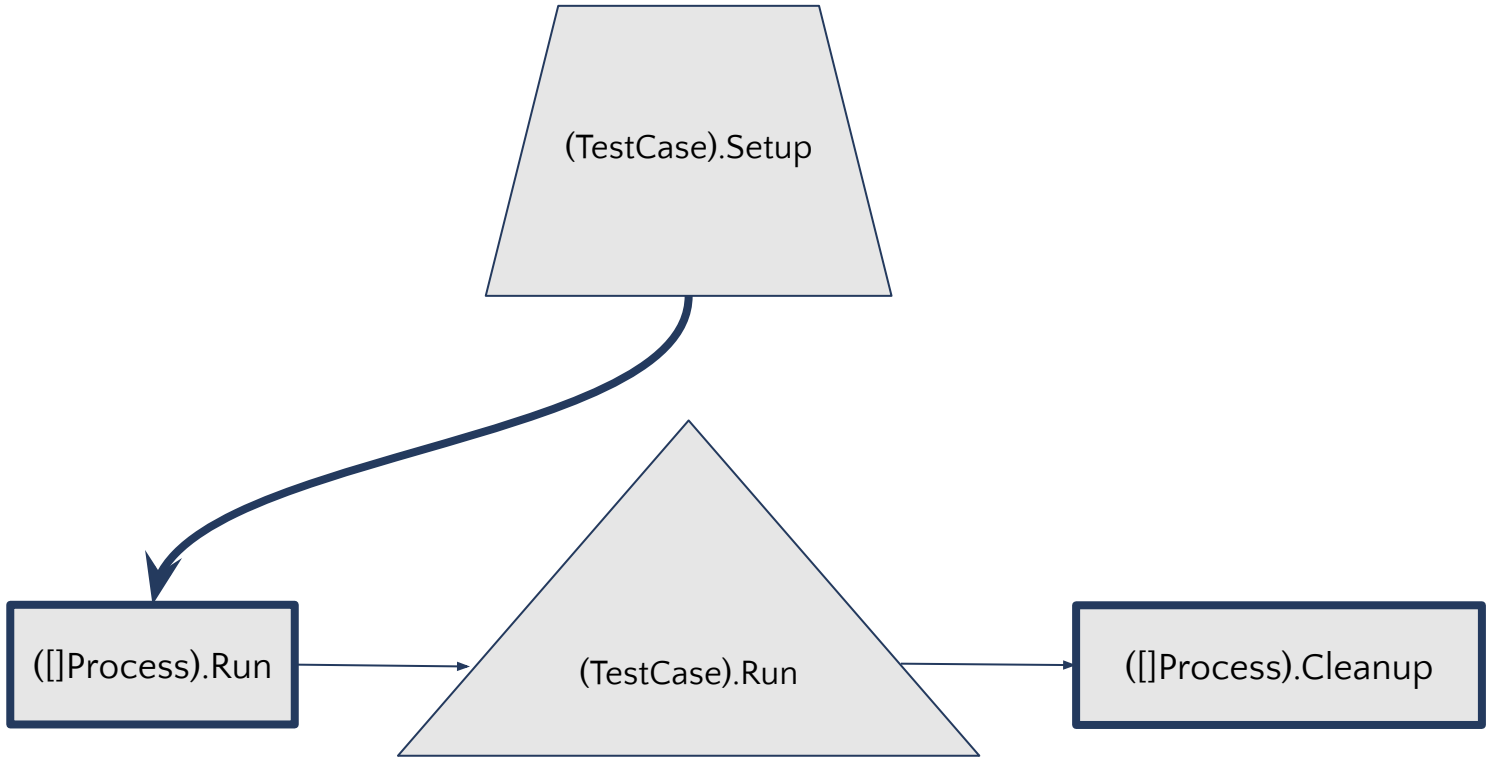
Test Case

```
func init() {
  suite.Register(new(base))
}

type base struct {
}

func (b *base) Setup(t *testing.T) []framework.Option {
  return []framework.Option{
    framework.WithProcesses(noop.New())
  }
}

func (b *base) Run(t *testing.T, ctx context.Context) {
  assert.Noop(t)
}
```



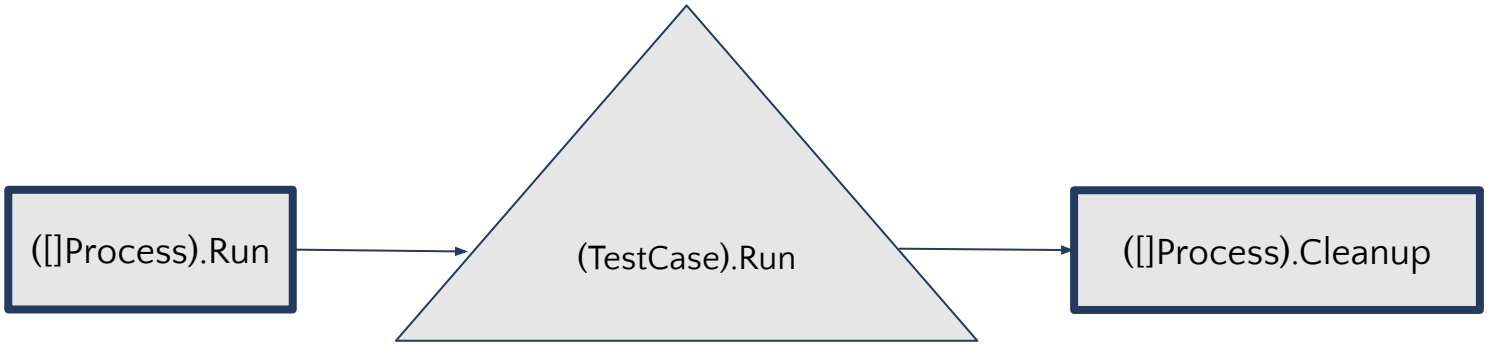


Framework Process



Framework Process

```go
// Interface is an interface for running and cleaning up a process.
type Interface interface {
  // Run runs the process.
  Run(*testing.T, context.Context)

  // Cleanup cleans up the process.
  Cleanup(*testing.T)
}
```



Process

```go
type NOOP struct {
  foo bool
}

func New(t *testing.T, fopts ...Option) *HTTP {
  t.Helper()

  var opts options
  for _, fopt := range fopts {
    fopt(&opts)
  }

  return &NOOP{
    foo: opts.foo,
  }
}

func (n *NOOP) Run(t *testing.T, ctx context.Context) {
  require.NoError(t,
    os.WriteFile(
      filepath.Join(t.TempDir(), "test.txt"),
      []byte("hello"),
      0600,
    ),
  )
}
```

```go
type Option func(*options)
```

```go
type options struct {
  foo bool
}

func WithFoo(foo bool) Option {
  return func(o *options) {
    o.foo = foo
  }
}
```

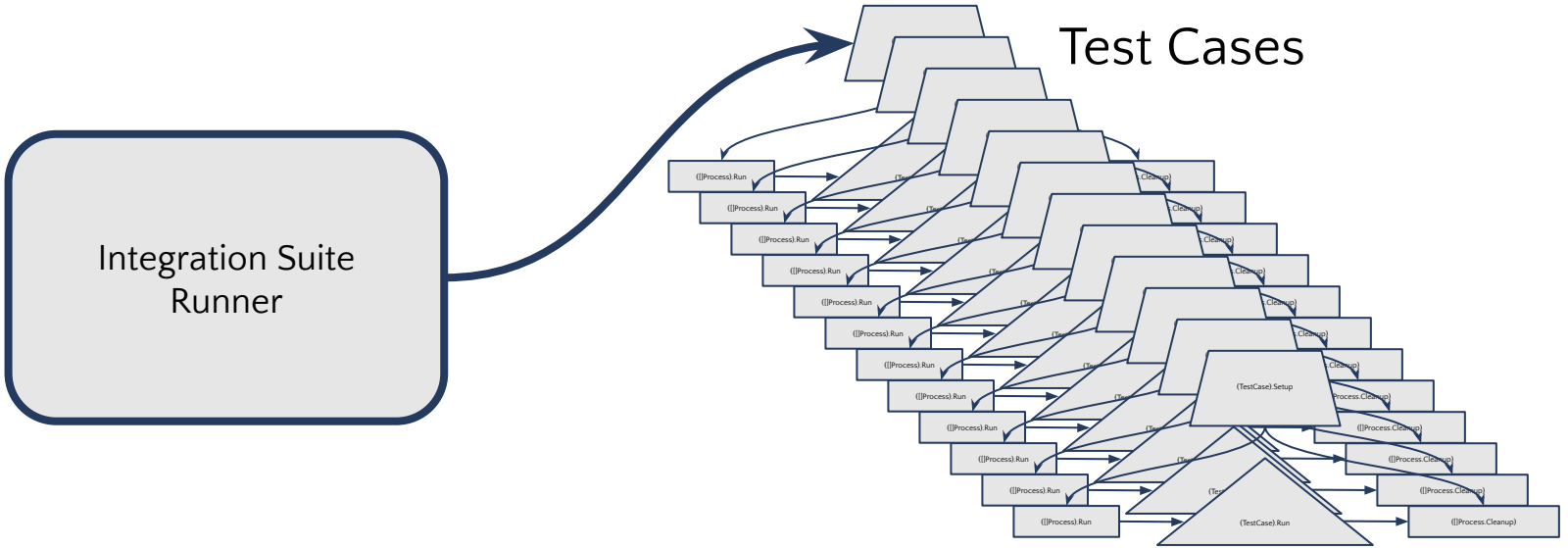
([]Process).Run

([]Process).Cleanup

Process – Run! ...

```
func Run(t *testing.T, ctx context.Context, opts ...Option) {
  t.Helper()

  o := options{}
  for _, opt := range opts {
    opt(&o)
  }

  t.Logf("starting %d processes", len(o.procs))

  for i, proc := range o.procs {
    i := i
    proc.Run(t, ctx)
    t.Cleanup(func() { o.procs[i].Cleanup(t) })
  }
}
```

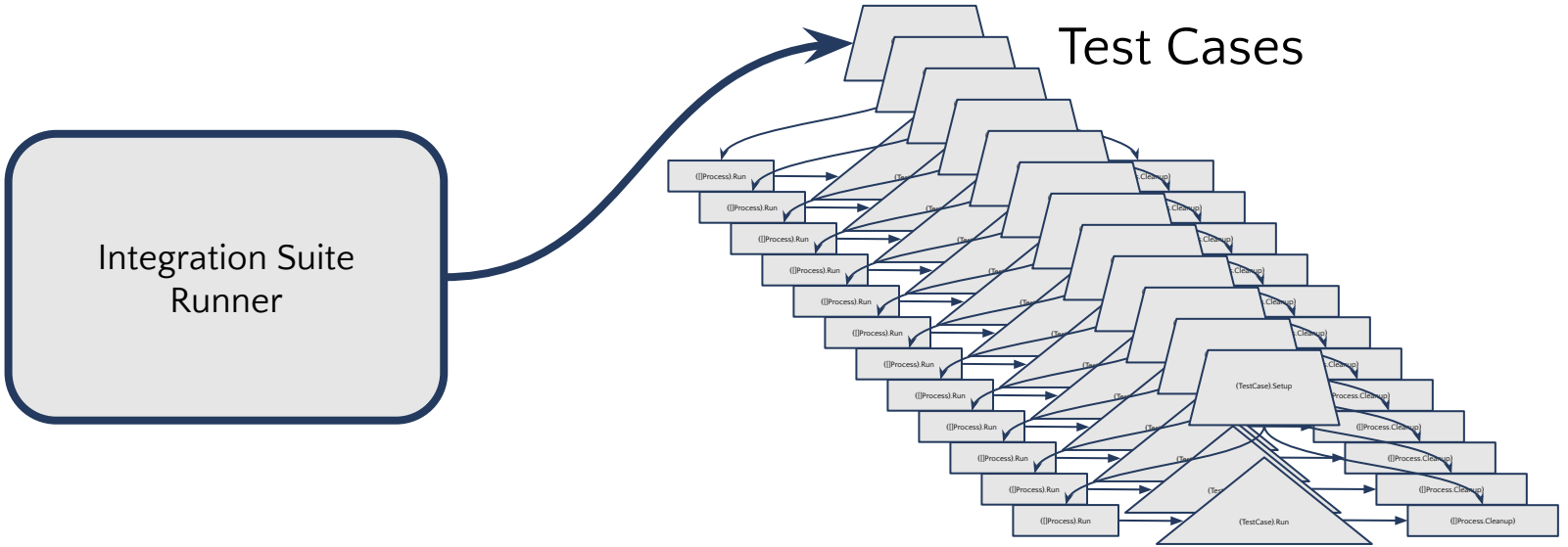
Integration Suite



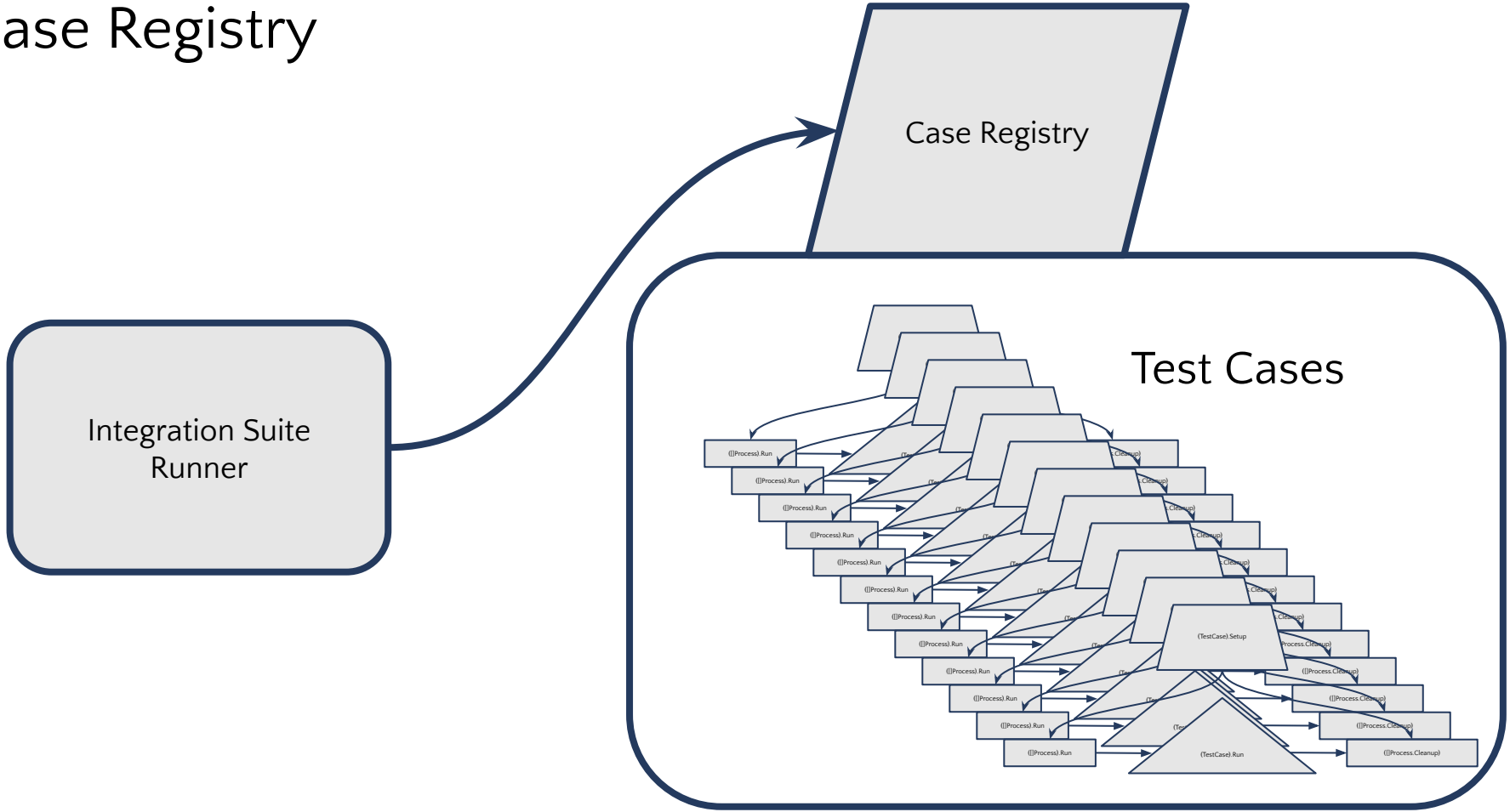
Integration Suite

```
func RunIntegrationTests(t *testing.T) {
  for _, tcase := range testSuite {
    t.Run(tcase.Name(), func(t *testing.T) {
      options := tcase.Setup(t)
      framework.Run(t, ctx, options...)
      tcase.Run(t, ctx)
    })
  }
}
```

Case Registry



Case Registry



Case Registry

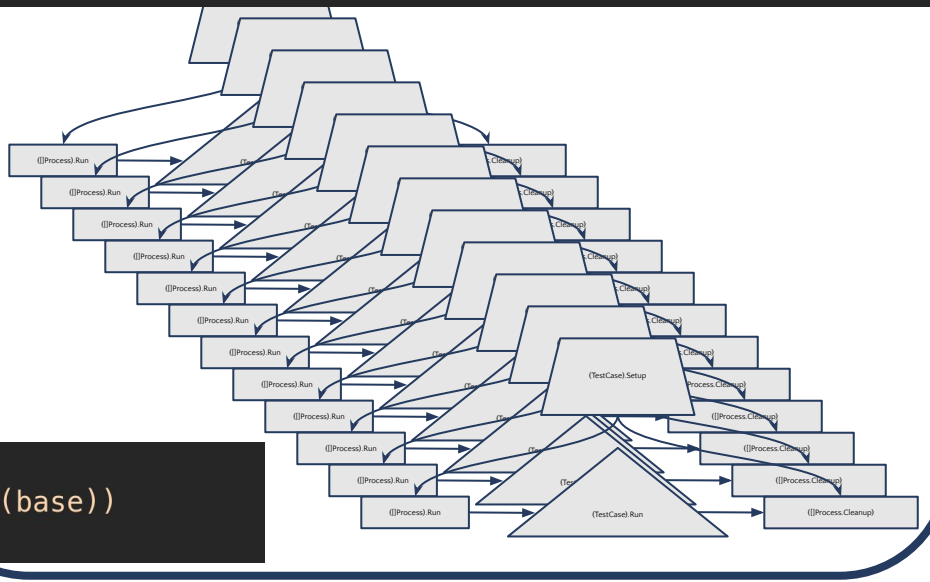
```
var cases []Case
```

Case Registry

```
import (
  _ "github.com/dapr/dapr/tests/integration/suite/daprd/base"
)
```

Integration Suite Runner

```
for _, tcase := range testSuite {
```



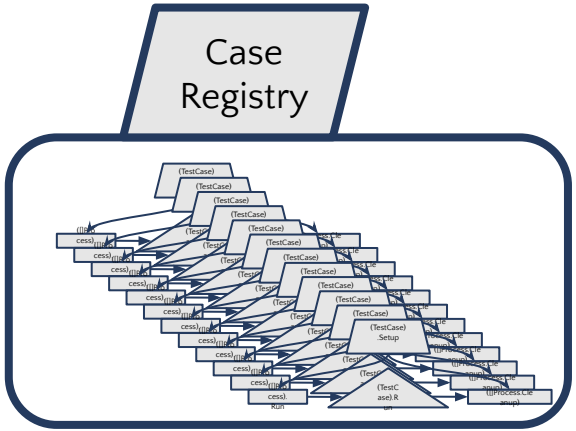

```
func init() {
  suite.Register(new(base))
}
```

NAMING (hard)

Test Naming

Follow idiomatic Go–

- Meaning derived through context
- Hierarchical
- Less is more
- lowercase, no underscores etc.

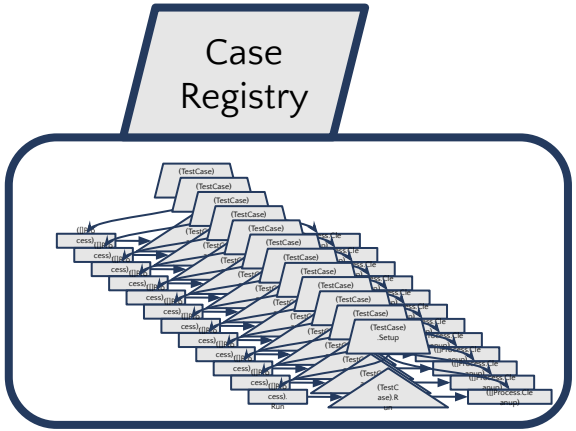


Test Naming

Follow idiomatic Go–

- Meaning derived through context
- Hierarchical
- Less is more
- lowercase, no underscore etc.

Reflect Magic!



Test Naming – Reflect Magic!

```go
// All returns all registered test cases.
// The returned slice is sorted by name.
func All(t *testing.T) []NamedCase {
  all := make([]NamedCase, len(cases))
  for i, tcase := range cases {
    tof := reflect.TypeOf(tcase).Elem()
    _, aft, ok := strings.Cut(tof.PkgPath(), "tests/integration/suite/")
    require.True(t, ok)
    name := aft + "/" + tof.Name()
    all[i] = NamedCase{name, tcase}
  }

  sort.Slice(all, func(i, j int) bool {
    return all[i].name < all[j].name
  })

  return all
}
```

Test Naming – Reflect Magic!

`TestName = packagePath + structName`

```go
// All returns all registered test cases.
// The returned slice is sorted by name.
func All(t *testing.T) []NamedCase {
  all := make([]NamedCase, len(cases))
  for i, tcase := range cases {
    tof := reflect.TypeOf(tcase).Elem()
    _, aft, ok := strings.Cut(tof.PkgPath(), "tests/integration/suite/")
    require.True(t, ok)
    name := aft + "/" + tof.Name()
    all[i] = NamedCase{name, tcase}
  }

  sort.Slice(all, func(i, j int) bool {
    return all[i].name < all[j].name
  })

  return all
}
```

```go
func init() {
  suite.Register(new(base))
}

type base struct {
}

func (b *base) Setup(t *testing.T) []framework.Option {
  return []framework.Option{
    framework.WithProcesses(noop.New())
  }
}

func (b *base) Run(t *testing.T, ctx context.Context) {
  assert.Noop(t)
}
```

`Test_Integration/daprd/foo/bas`
e

Test Naming – Why?

Test Naming – Focus!

```
$ go test --focus "actors|placement"
```

Test Naming - Focus with regexp

```
$ go test --focus "actors|placement"
```

```
focusedTests := make([]suite.NamedCase, 0)
focus, err := regexp.Compile(*focusF)
require.NoError(t, err)
for _, tcase := range suite.All(t) {
  if !focus.MatchString(tcase.Name()) {
    continue
  }
  focusedTests = append(focusedTests, tcase)
}
```

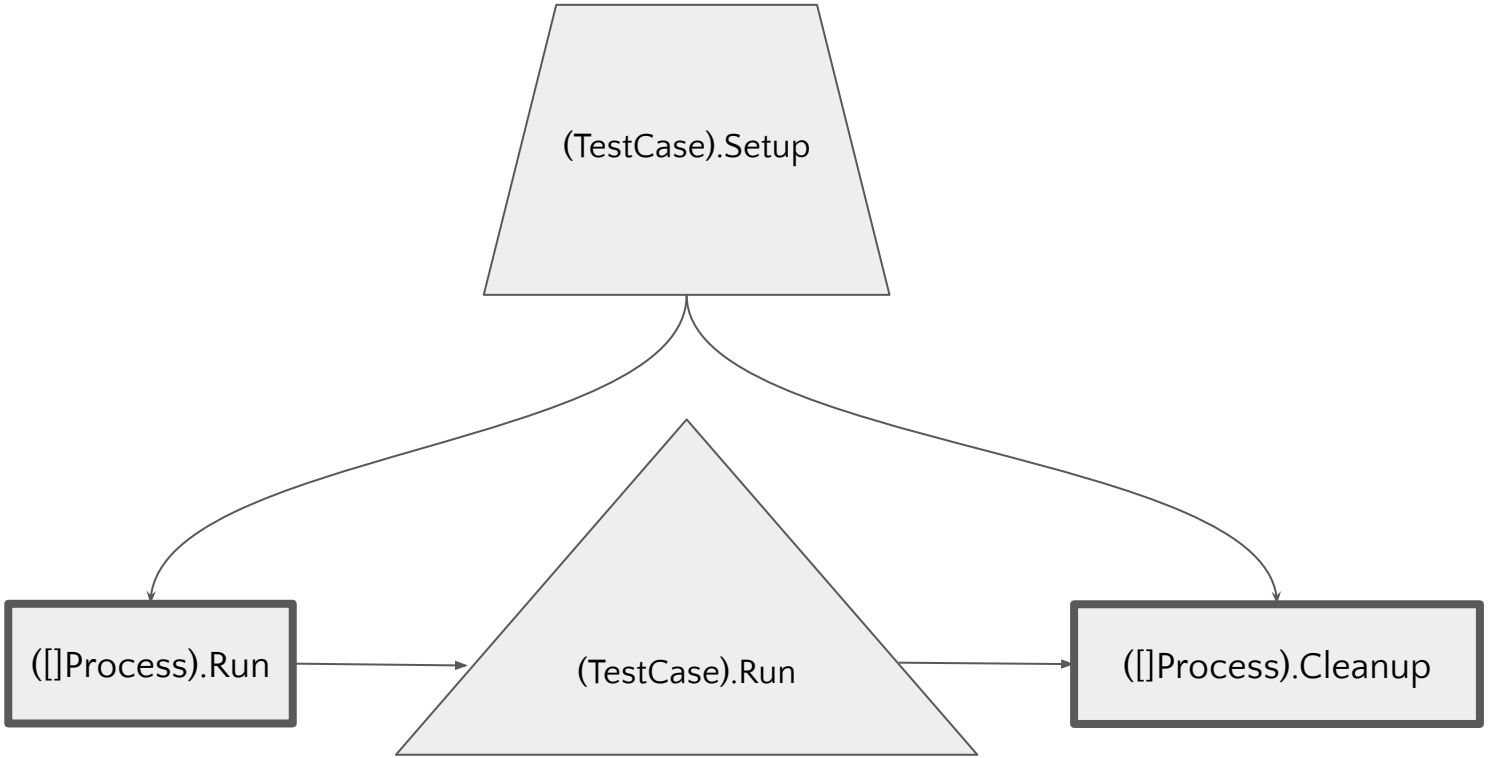
Test Naming – Focus!

```
$ go test -v -focus sentry
```

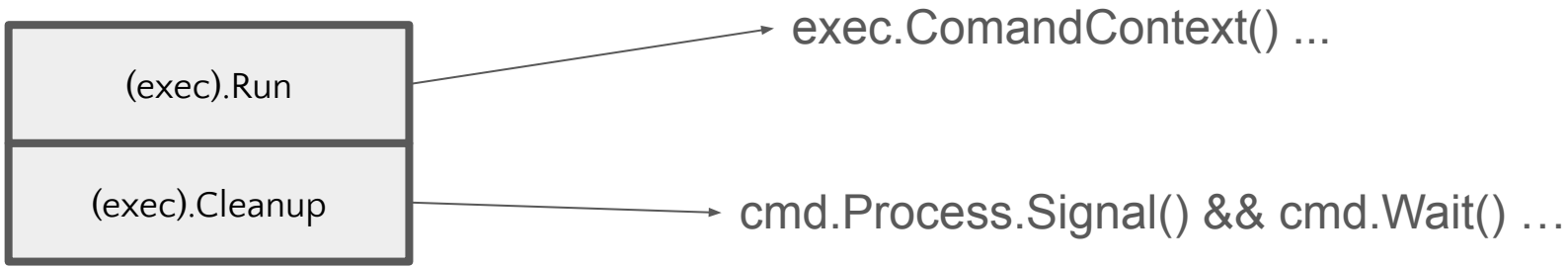
```
=== NAME  Test_Integration
    integration.go:77: Total integration test execution time: 3.9s
--- PASS: Test_Integration (4.95s)
    --- PASS: Test_Integration/build_binaries (1.00s)
    --- PASS: Test_Integration/healthz/sentry (0.23s)
        --- PASS: Test_Integration/healthz/sentry/run (0.21s)
    --- PASS: Test_Integration/ports/sentry (0.32s)
        --- PASS: Test_Integration/ports/sentry/run (0.31s)
    --- PASS: Test_Integration/sentry/metrics/expiry (0.27s)
        --- PASS: Test_Integration/sentry/metrics/expiry/run (0.22s)
    --- PASS: Test_Integration/sentry/validator/insecure/insecure (0.14s)
        --- PASS: Test_Integration/sentry/validator/insecure/insecure/run (0.12s)
            --- PASS: Test_Integration/sentry/validator/insecure/insecure/run/fails_when_passing_an_invalid_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/insecure/insecure/run/issue_a_certificate_with_insecure_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/insecure/insecure/run/insecure_validator_is_the_default (0.00s)
            --- PASS: Test_Integration/sentry/validator/insecure/insecure/run/fails_with_missing_CSR (0.00s)
            --- PASS: Test_Integration/sentry/validator/insecure/insecure/run/fails_with_missing_namespace (0.00s)
            --- PASS: Test_Integration/sentry/validator/insecure/insecure/run/fails_with_invalid_CSR (0.00s)
    --- PASS: Test_Integration/sentry/validator/jwks/httpsCA (0.13s)
        --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run (0.12s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_passing_an_invalid_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_passing_the_insecure_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_no_validator_is_passed (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/issue_a_certificate_with_JWKS_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_token_has_invalid_audience (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_token_has_invalid_subject (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_token_is_expired (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_when_token_is_not_yet_valid (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/httpsCA/run/fails_with_token_signed_by_wrong_key (0.00s)
    --- PASS: Test_Integration/sentry/validator/jwks/jwks (0.13s)
        --- PASS: Test_Integration/sentry/validator/jwks/jwks/run (0.12s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_passing_an_invalid_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_passing_the_insecure_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_no_validator_is_passed (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/issue_a_certificate_with_JWKS_validator (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_token_has_invalid_audience (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_token_has_invalid_subject (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_token_is_expired (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_when_token_is_not_yet_valid (0.00s)
            --- PASS: Test_Integration/sentry/validator/jwks/jwks/run/fails_with_token_signed_by_wrong_key (0.00s)
    --- PASS: Test_Integration/sentry/validator/kubernetes/kubernetes (1.38s)
        --- PASS: Test_Integration/sentry/validator/kubernetes/kubernetes/run (0.21s)
    --- PASS: Test_Integration/sentry/validator/kubernetes/longname (1.32s)
        --- PASS: Test_Integration/sentry/validator/kubernetes/longname/run (0.22s)
PASS
ok      github.com/dapr/dapr/tests/integration  6.034s
```

PROCESS - w(rap)

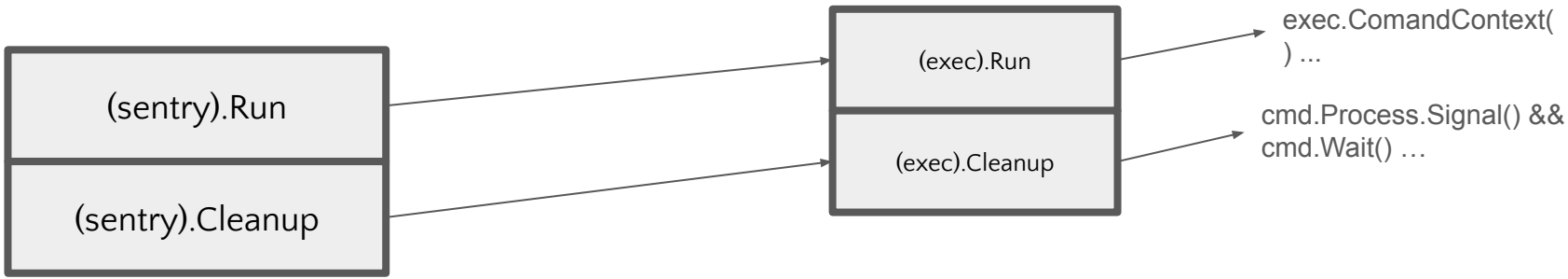
Process - w(rap)



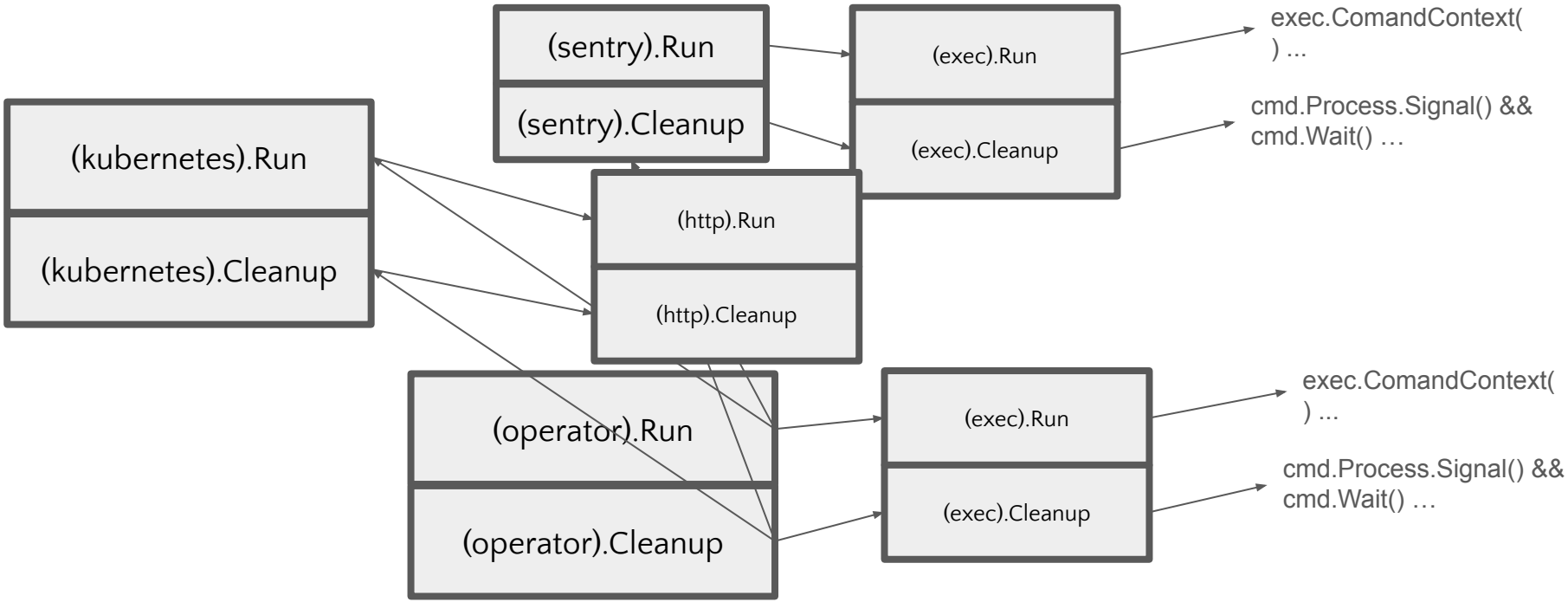
Process - w(rap) - exec



Process - w(rap) - Unix Philosophy



Process - w(rap) - Unix Philosophy



Process - w(rap) - Unix Philosophy

```
return []framework.Option{
  framework.WithProcesses(sentry,
    w.operatorCreate, w.operatorUpdate, w.operatorDelete,
    w.loglineCreate, w.loglineUpdate, w.loglineDelete,
    w.daprdCreate, w.daprdUpdate, w.daprdDelete,
  ),
}
```

```
func Run(t *testing.T, ctx context.Context, opts ...Option) {
  t.Helper()

  o := options{}
  for _, opt := range opts {
    opt(&o)
  }

  t.Logf("starting %d processes", len(o.procs))

  for i, proc := range o.procs {
    i := i
    proc.Run(t, ctx)
    t.Cleanup(func() { o.procs[i].Cleanup(t) })
  }
}
```

PROCESS - bin

Build from Source

Build from Source

Go’s build cache is very powerful.

Build from Source

```
func Build(t *testing.T, name string) {
  t.Helper()
  if _, ok := os.LookupEnv(EnvKey(name)); !ok {
    t.Logf("%q not set, building %q binary", EnvKey(name), name)

    _, tfile, _, ok := runtime.Caller(0)
    require.True(t, ok)
    rootDir := filepath.Join(filepath.Dir(tfile), "../../../..")

    // Use a consistent temp dir for the binary so that the binary is cached on
    // subsequent runs.
    binPath := filepath.Join(os.TempDir(), "dapr_integration_tests/"+name)
    if runtime.GOOS == "windows" {
      binPath += ".exe"
    }

    ioout := iowriter.New(t, name)
    ioerr := iowriter.New(t, name)

    t.Logf("Root dir: %q", rootDir)
    t.Logf("Compiling %q binary to: %q", name, binPath)
    cmd := exec.Command("go", "build", "-tags=allcomponents", "-v", "-o", binPath,
)
    cmd.Dir = rootDir
    cmd.Stdout = ioout
    cmd.Stderr = ioerr
    // Ensure CGO is disabled to avoid linking against system libraries.
    cmd.Env = append(os.Environ(), "CGO_ENABLED=0")
    require.NoError(t, cmd.Run())

    require.NoError(t, ioout.Close())
```

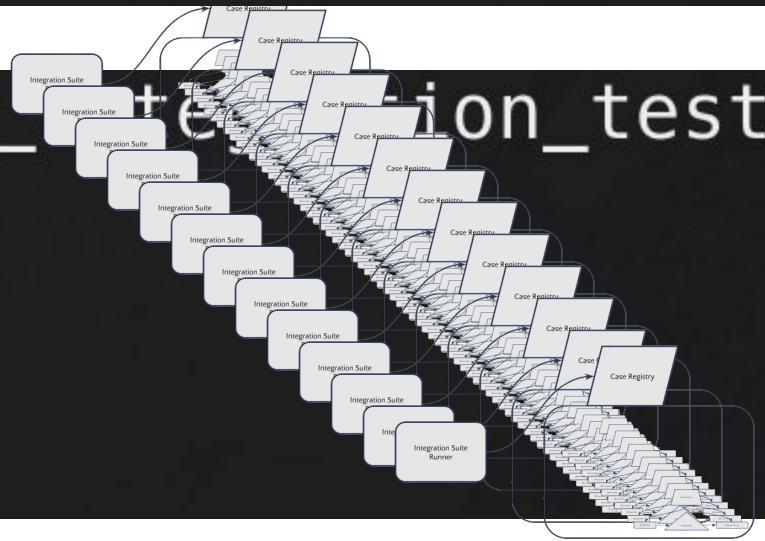
Build from Source- Go Cache Magic!

```
$ ls -1 /tmp/dapr_integration_tests
daprd
operator
placement
sentry
```

Build from Source- Go Cache Magic!

```
$ go test -v -focus sentr
```

```
$ ls -1 /tmp/dapr_te       ion_tests
daprd
operator
placement
sentry
```



Version Skew Tests

p|pe

Process – P|pe

- Software writes messages to “logs”
- These can be noisy
- This fills disk space
- This makes it impossible to read test output

Process - P|pe

Capture exec pipes to in-memory buffers

```
stdout: iowriter.New(t, filepath.Base(binPath)),
stderr: iowriter.New(t, filepath.Base(binPath)),
```

Process - P|pe

Only write logs to test output when it matters- when the test fails!

```go
// flush writes the buffer to the test logger. Expects the lock to be held
// before calling.
func (w *stdwriter) flush() {
  w.lock.Lock()
  defer w.lock.Unlock()
  defer w.buf.Reset()

  // Don't log if the test hasn't failed and the user hasn't requested logs to
  // always be printed.
  if !w.t.Failed() &&
    !utils.IsTruthy(os.Getenv("DAPR_INTEGRATION_LOGS")) {
    return
  }

  for {
    line, err := w.buf.ReadBytes('\n')
    if len(line) > 0 {
      w.t.Log(w.t.Name() + "/" + w.procName + ": " +
        strings.TrimSuffix(string(line), "\n"))
    }
    if err != nil {
      if !errors.Is(err, io.EOF) {
        w.t.Log(w.t.Name() + "/" + w.procName + ": " + err.Error())
      }
      break
    }
  }
}
```

Process - P|pe

You can even test against the output!
(logline is yet another process)

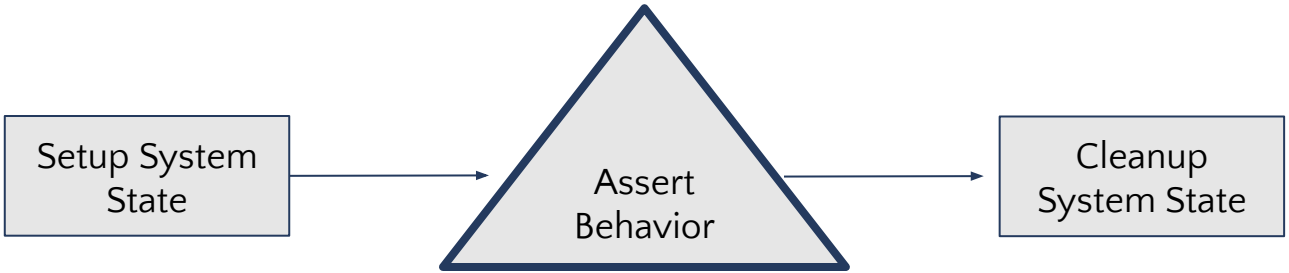
```
i.logline = logline.New(t,
  logline.WithStdoutLineContains(
    "Blocking graceful shutdown for 2s or until app reports unhealthy...",
    "Block shutdown period expired, entering shutdown...",
    "Daprd shutdown gracefully",
  ),
)
```

```
    daprd.WithExecOptions(exec.WithStdout(i.logline.Stdout())),
```

Assert eventually

Assert eventually

- All software is eventually consistent
- Asserting behaviour requires waiting for it to complete
- We **have** to wait for that process to complete to observe it, however
- **Never** use `time.Sleep` …



Assert eventually

`time.Sleep` is nuclear

- If a single test sleeps for 5 seconds
- CI runs 4 times a day
- This equates to **2 hours** of idle CPU a year...

- Dapr (currently) has 133 integration tests
- If just 10% of those tests Sleep for 5 seconds
- This equates to **more than an entire day** of idle CPU a year...

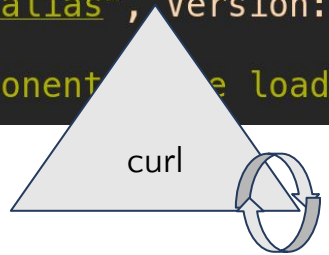
- Think of the polar bears (and developers...)

Assert eventually

- Use polling (eventually) with short intervals to assert behaviour
- testify is your friend

```
"github.com/stretchr/testify/assert"
"github.com/stretchr/testify/require"
```

```
assert.EventuallyWithT(t, func(c *assert.CollectT) {
  resp := util.GetMetaComponents(c, ctx, client, u.daprd1.HTTPPort())
  assert.ElementsMatch(c, []*rtv1.RegisteredComponents{
    {Name: "uppercase", Type: "middleware.http.routeralias", Version: "v1"},
    {Name: "uppercase2", Type: "middleware.http.routeralias", Version: "v1"},
  }, resp)
}, time.Second*5, time.Millisecond*100, "expected component[redacted] loaded")
```



CLEANUP (really)

CLEANUP (really)

Tests should never leak → Cleanup System State

- As test case number increases, so too could resource consumption
- Every test should be given a clean “sandbox” to run in

- Leaking state between tests gives breaks → Setup System State

CLEANUP (really)

Some of your friends include:

- `t.TempDir()`
- `t.Cleanup()`
- `Port 0`
- `In-Memory state stores/sqlite`
- `No dependency on the Internet`
- `cmd.Wait()`
- `Functions should not receive stop channels`
- ❤️ `context.Context` ❤️

OS

(i use nixos btw)

OS - oh cool, I use NixOS actually

Some operating systems are *weird*

Use build flags where you can, and work through the pain..

```
//go:build !windows
// +build !windows
```

```
//go:build windows
// +build windows
```

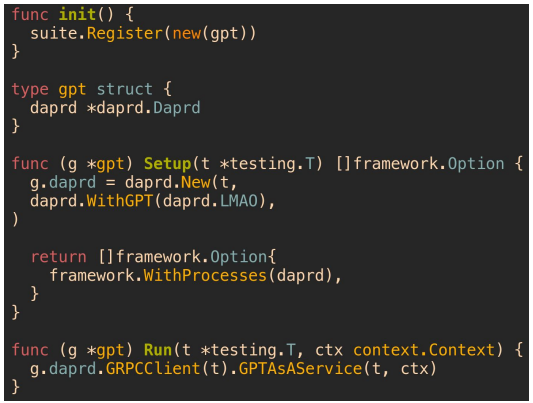
```
if runtime.GOOS == "windows" {
  binPath += ".exe"
}
```

```
if runtime.GOOS == "windows" {
  return !strings.Contains(err.Error(), "An existing connection was forcibly closed by the remote host.")
}
```

Being Productive

Being Productive

- Building a culture of integration testing in a distributed team is always a WIP
- A good testing framework should be usable as a feature development sandbox
- The more higher-order your Processes are the more productive writing tests (and features/experiments) your team will be



```
func init() {
  suite.Register(new(gpt))
}

type gpt struct {
  daprd *daprd.Daprd
}

func (g *gpt) Setup(t *testing.T) []framework.Option {
  g.daprd = daprd.New(t,
  daprd.WithGPT(daprd.LMAO),
)

  return []framework.Option{
    framework.WithProcesses(daprd),
  }
}

func (g *gpt) Run(t *testing.T, ctx context.Context) {
  g.daprd.GRPCClient(t).GPTAsAService(t, ctx)
}
```

Efficient Integration Testing in Go

A Case Study on Dapr

Josh van Leeuwen