BUOYANT

The creators of LINKERD



What really happens at startup

A deep dive into Linkerd, init containers, CNI plugins, and more

Flynn, Tech Evangelist
flynn@buoyant.io | @flynn on the Linkerd Slack



What's on the agenda?



Today is more slide-heavy than usual.

- Kubernetes pod startup is complex.
- We need to cover that *plus* how Linkerd fits into it…
- …and there are multiple paths we need to talk about.
- We will still do some hands-on work though!

What's needed for the hands-on part?



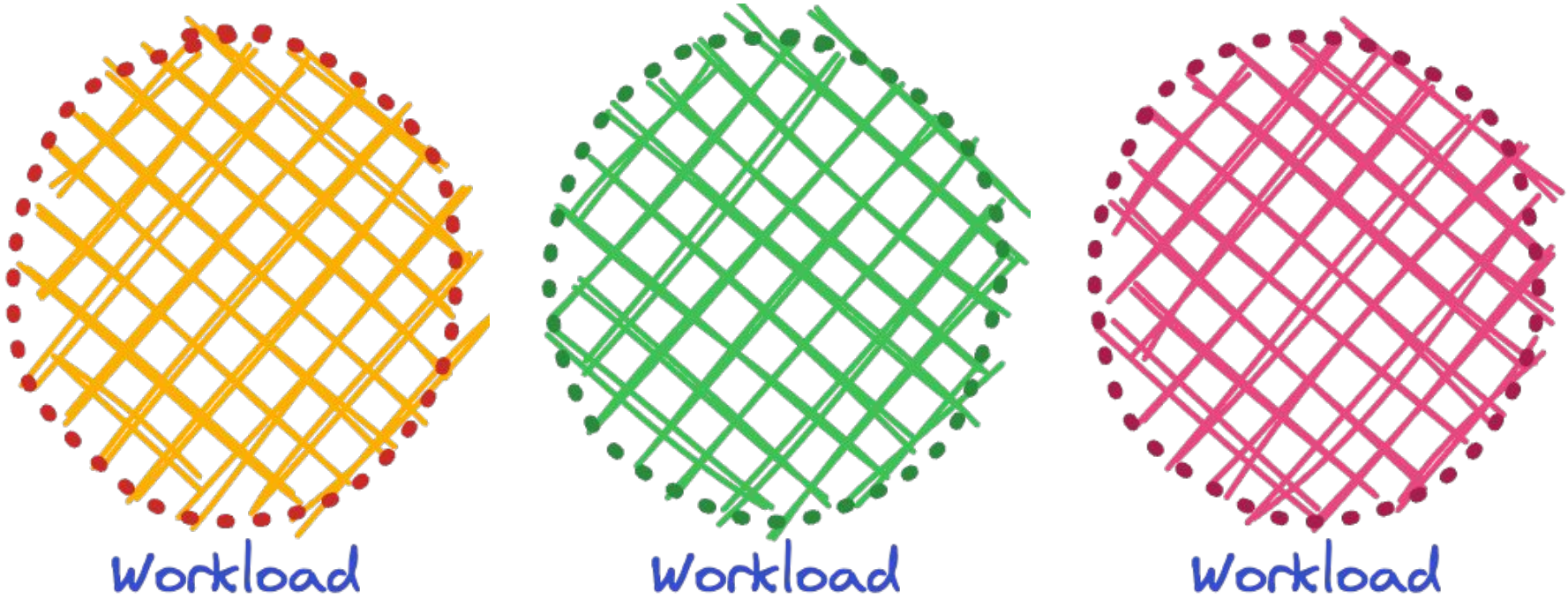
Today, what kind of cluster you use matters.

- ➔ This is because we're messing with the CNI, and the CNI depends on what kind of cluster you have.
- ➔ To follow along, you'll need `kubectl` and `linkerd`, but also `k3d`:

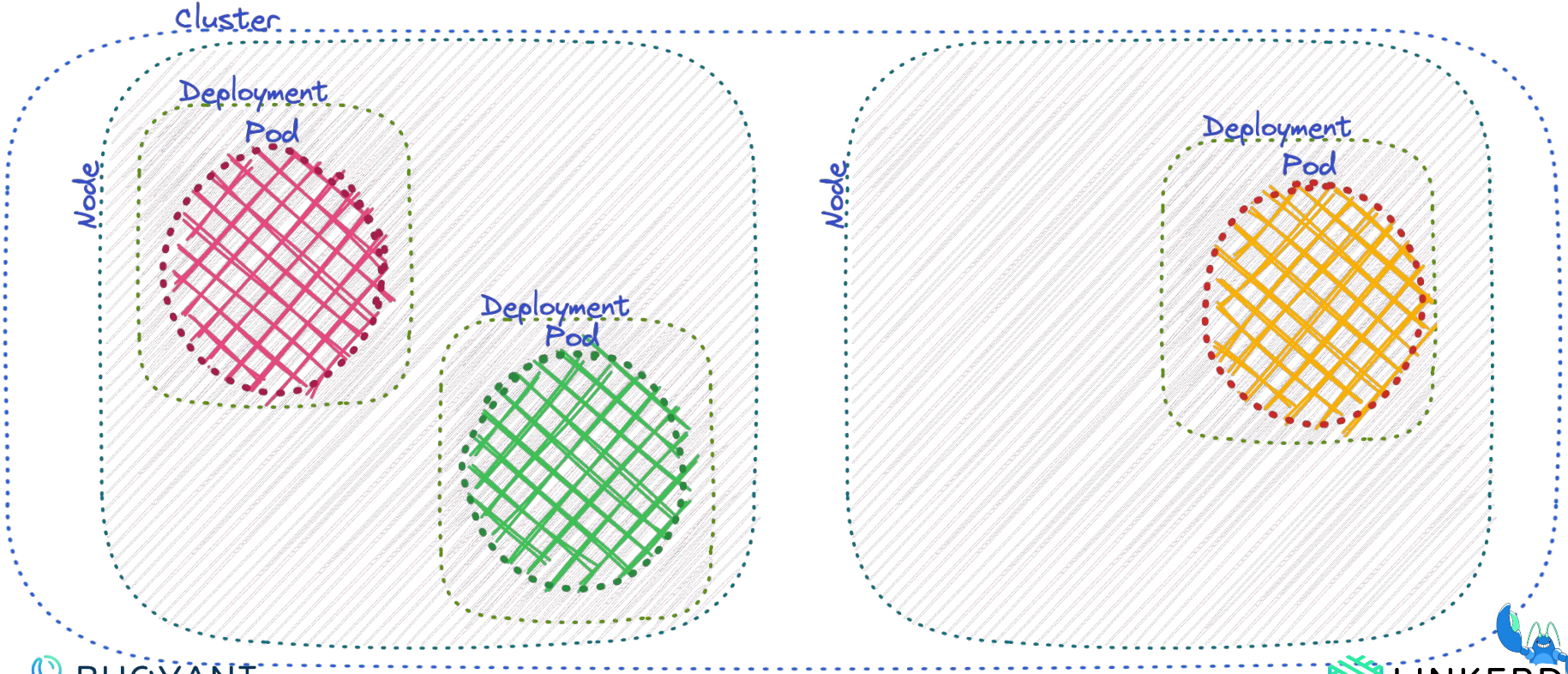
  `https://k3d.io/v5.4.6/#installation`

  or `brew install k3d` on a Mac

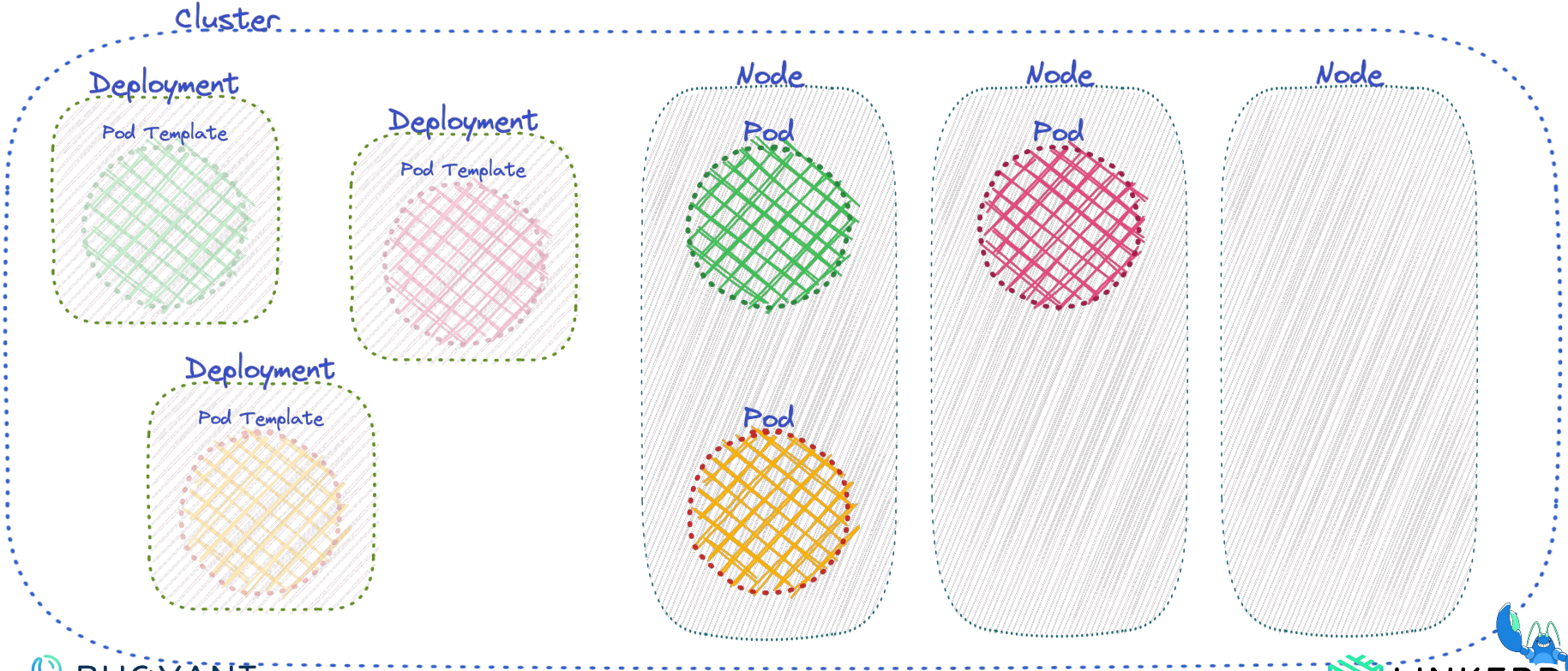
Kubernetes Overgeneralized



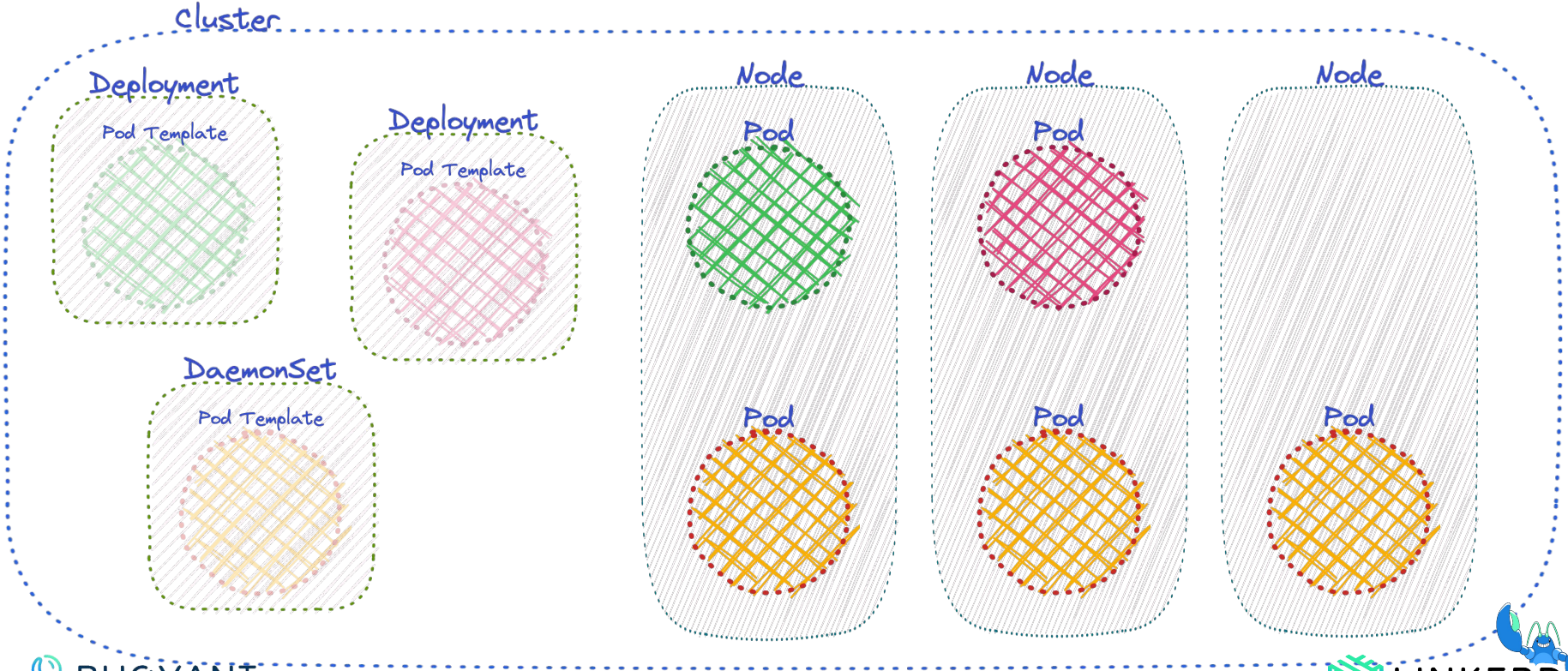
Kubernetes Overgeneralized



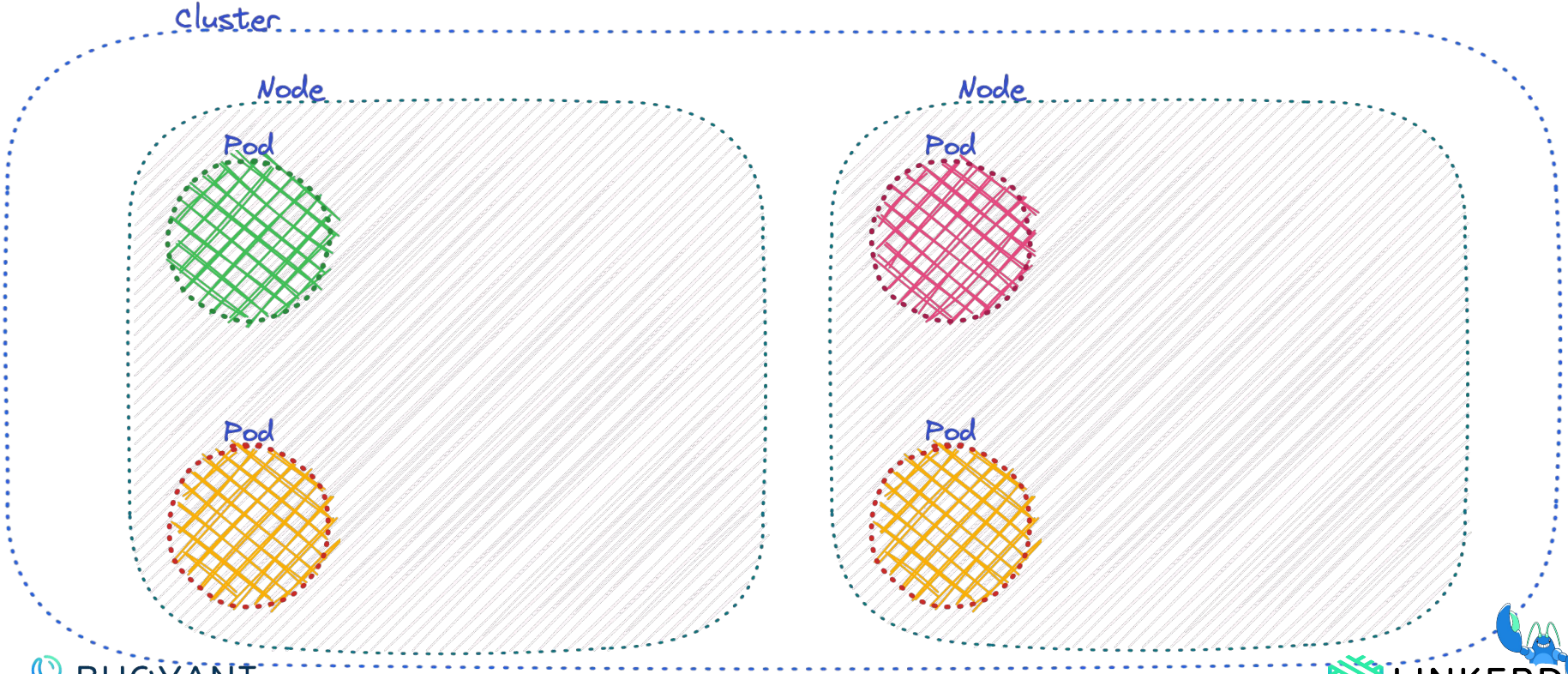
Down the Rabbit Hole



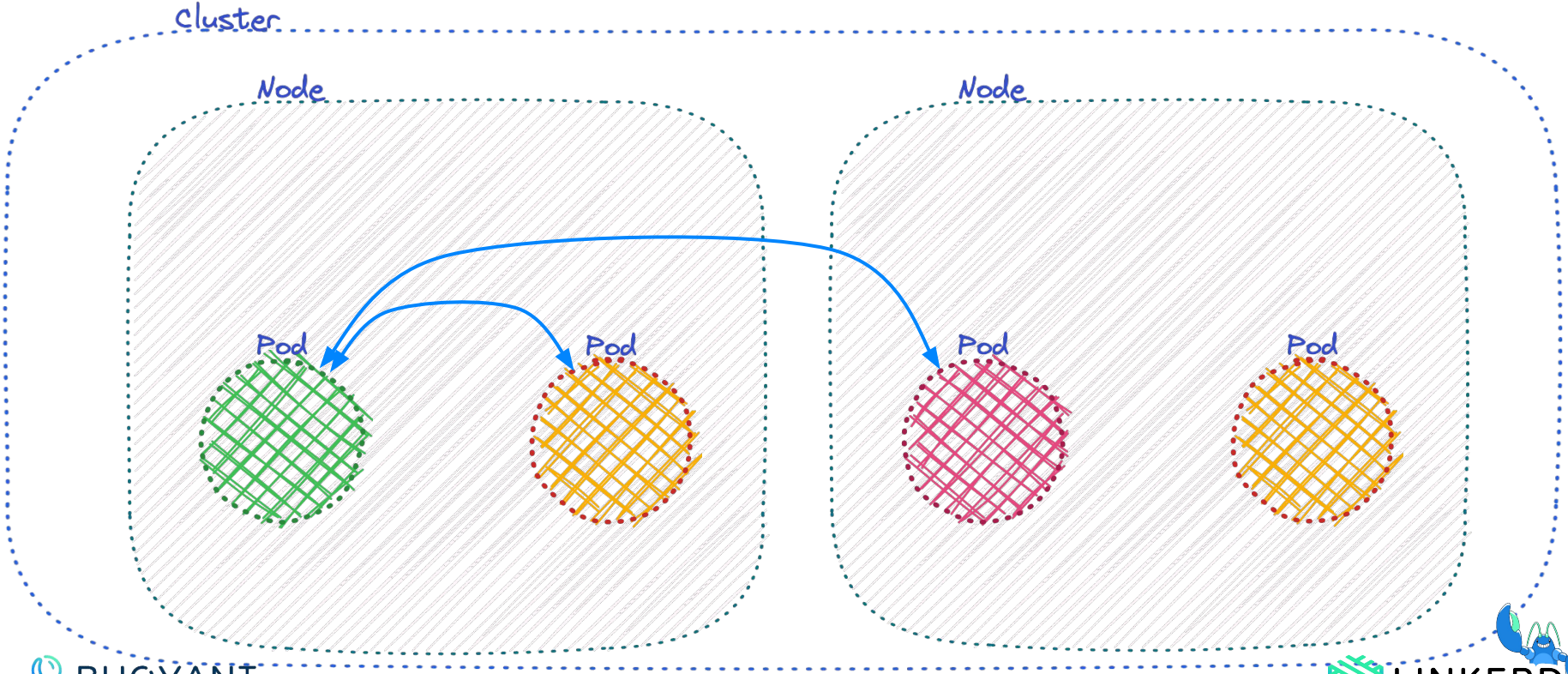
Down the Rabbit Hole



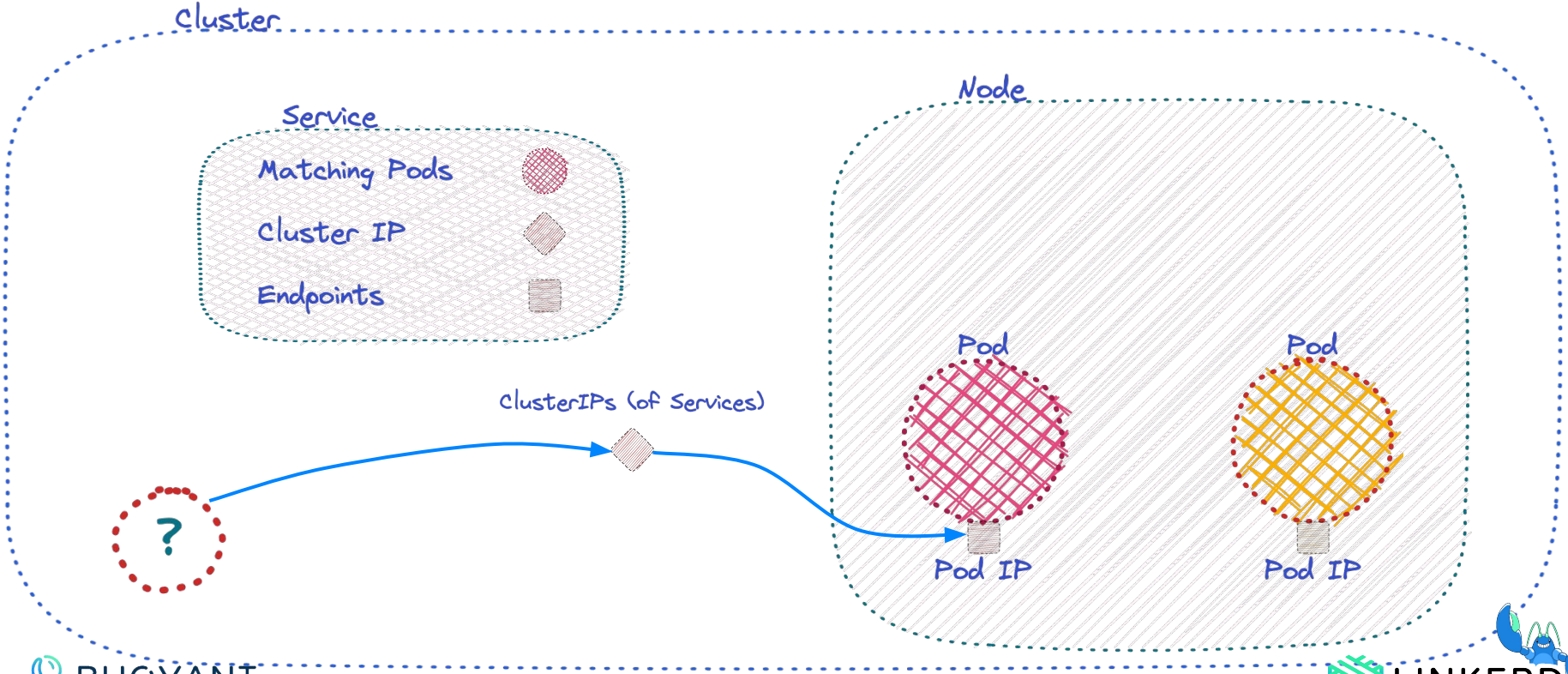
Down the Rabbit Hole



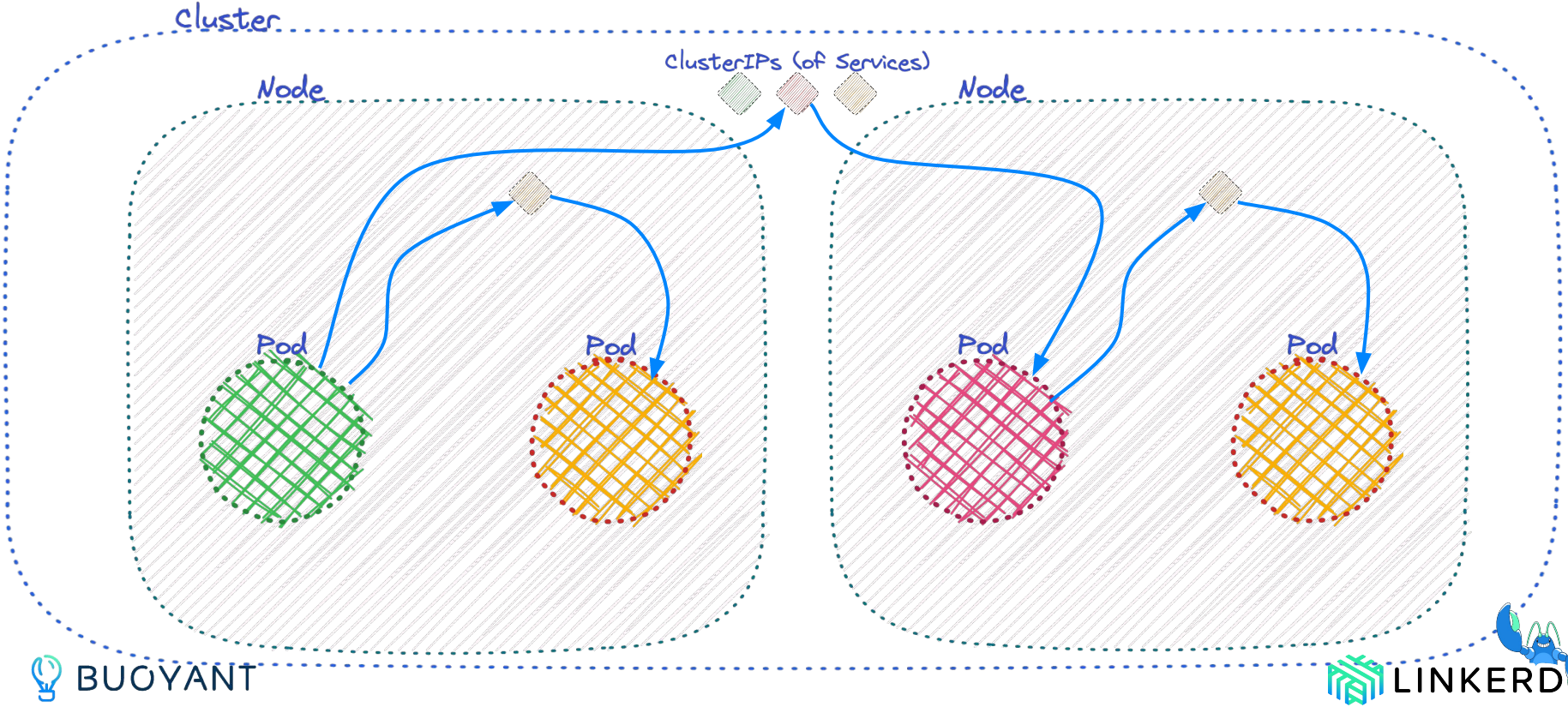
Down the Rabbit Hole



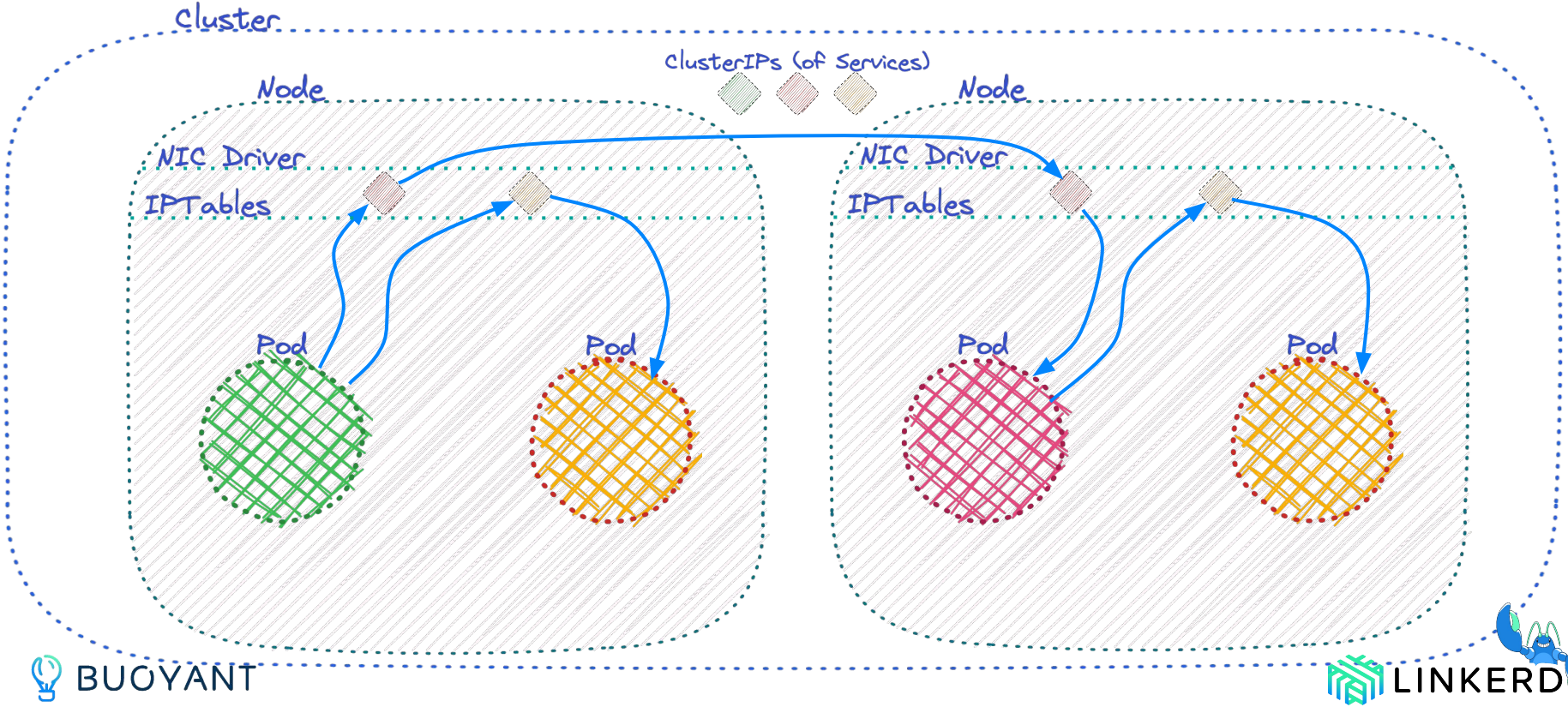
Down the Rabbit Hole



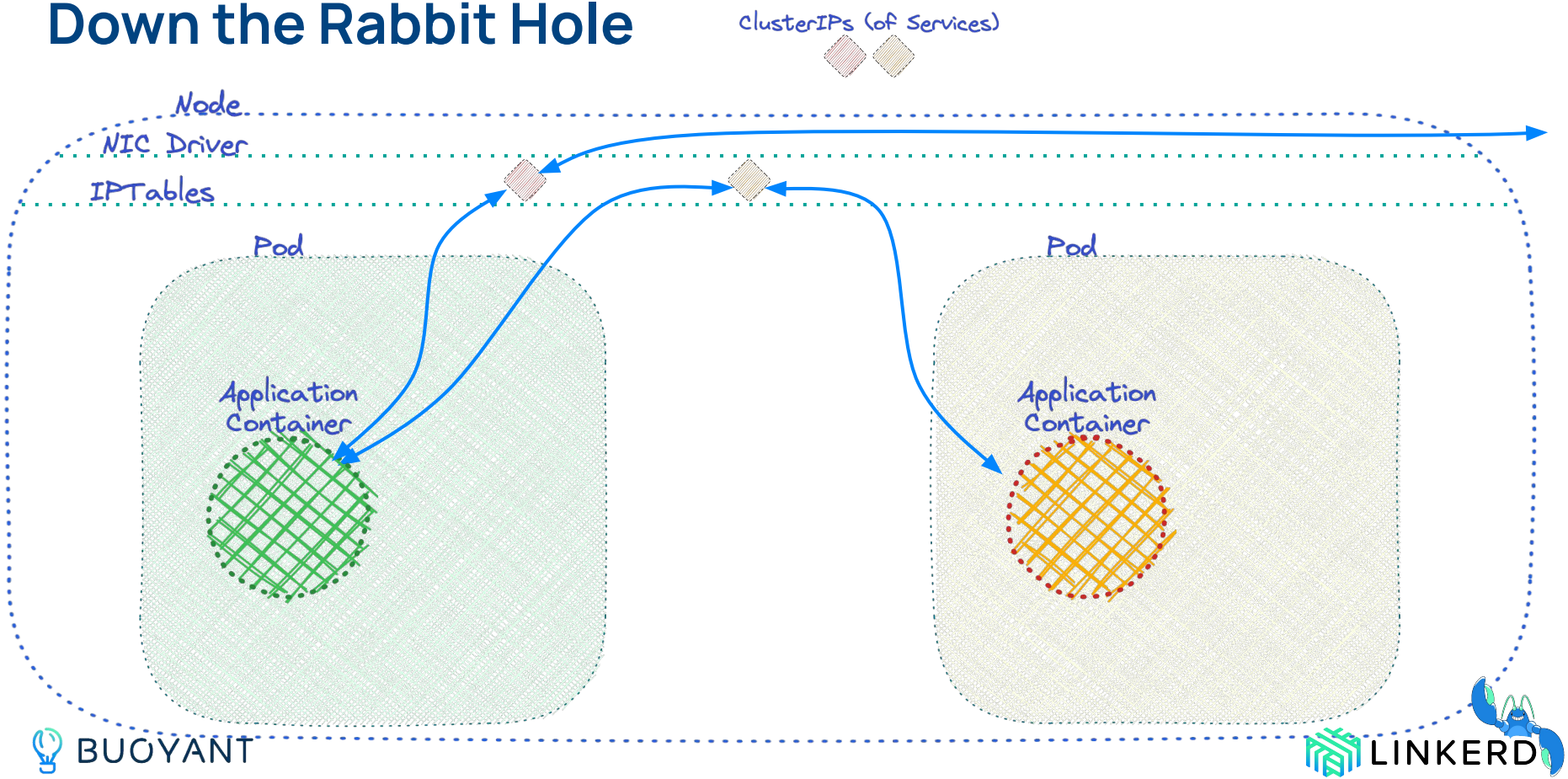
Down the Rabbit Hole



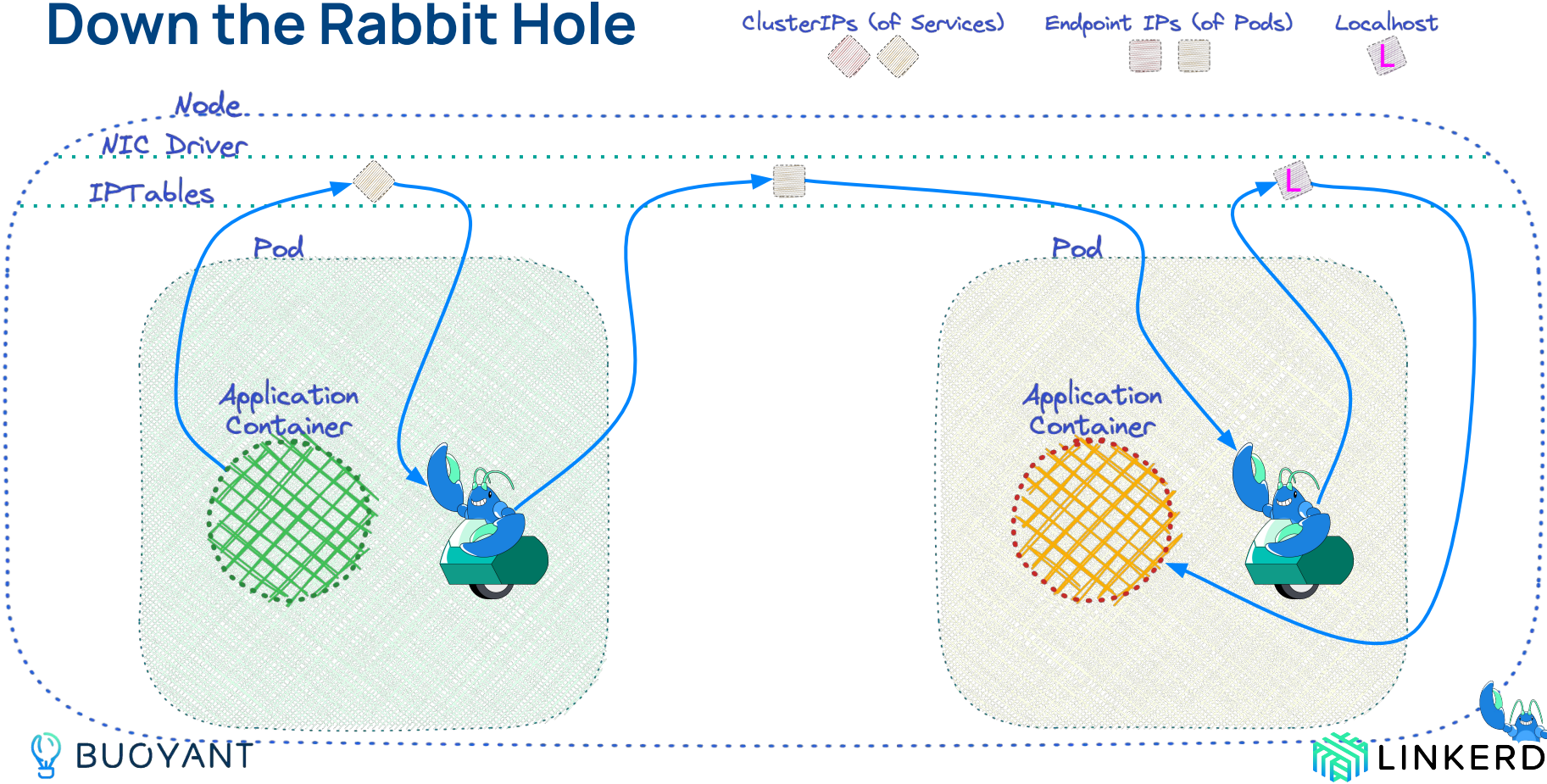
Down the Rabbit Hole



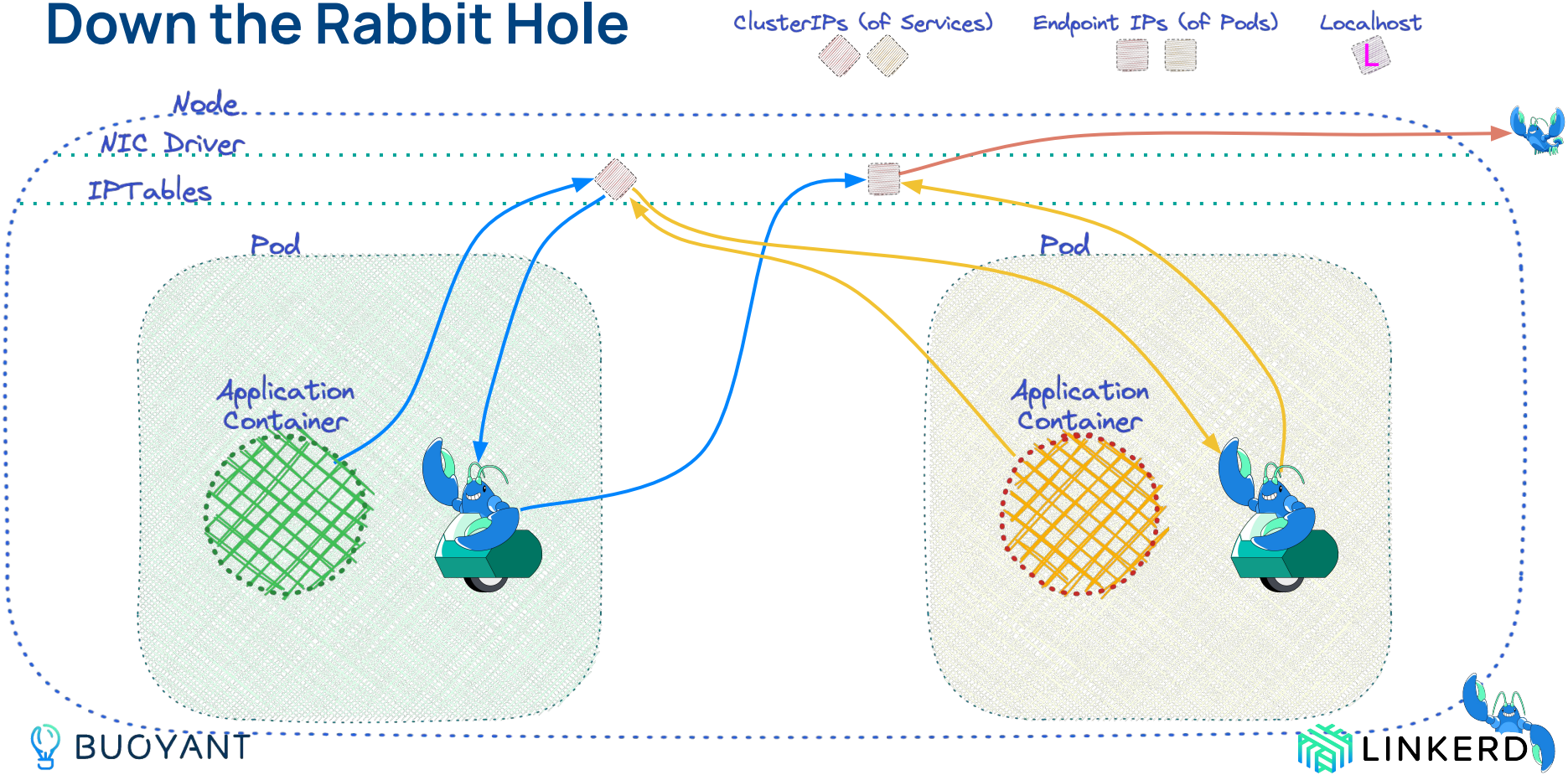
Down the Rabbit Hole



Down the Rabbit Hole



Down the Rabbit Hole



BUOYANT

The creators of LINKERD

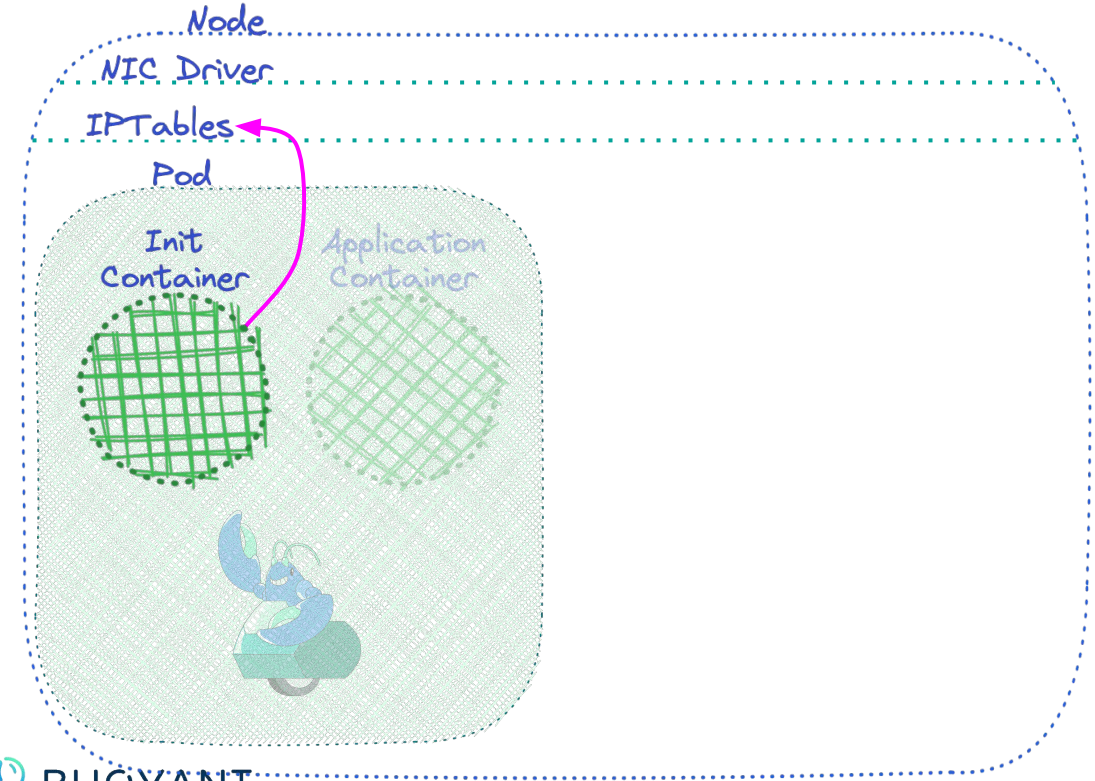
How is all this configured??!



Startup, Containers, and Plugins, oh my!

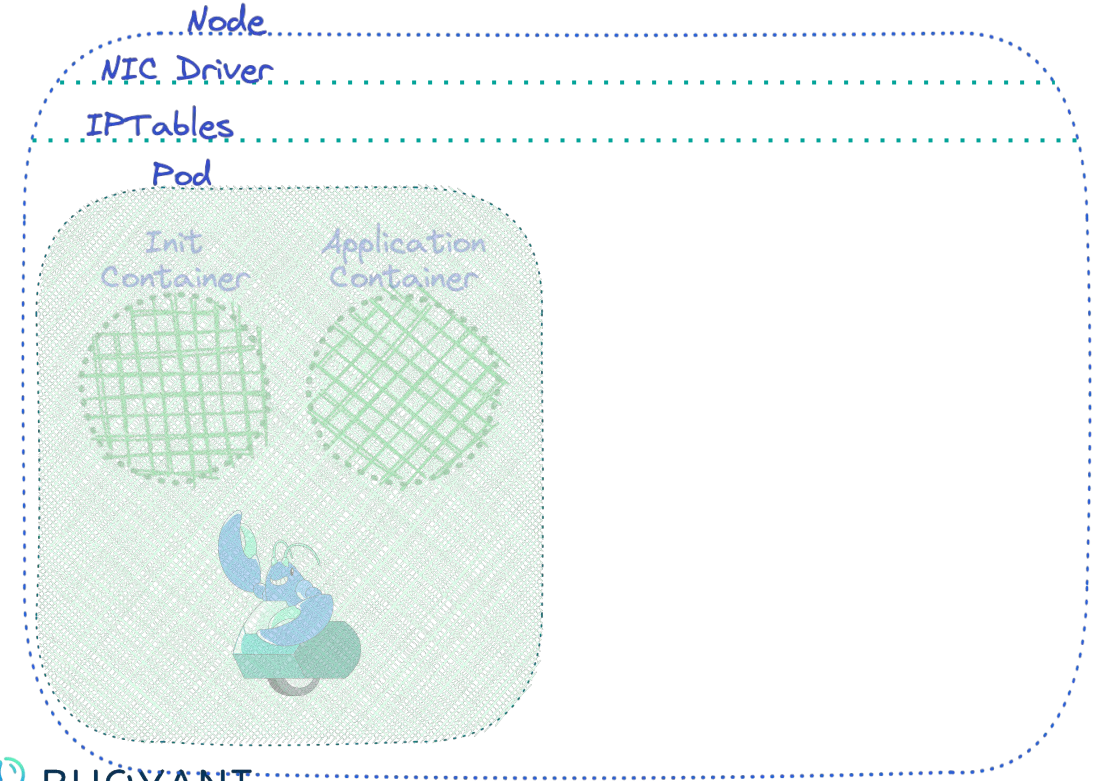
- Two possibilities for configuring all this networking stuff:
  - Kubernetes *init container*
  - Kubernetes *CNI plugin*
- Also multiple paths through the two methods!
  - One happy path 🙂
  - Multiple *less*-happy paths 😕

Init Container: Happy Path



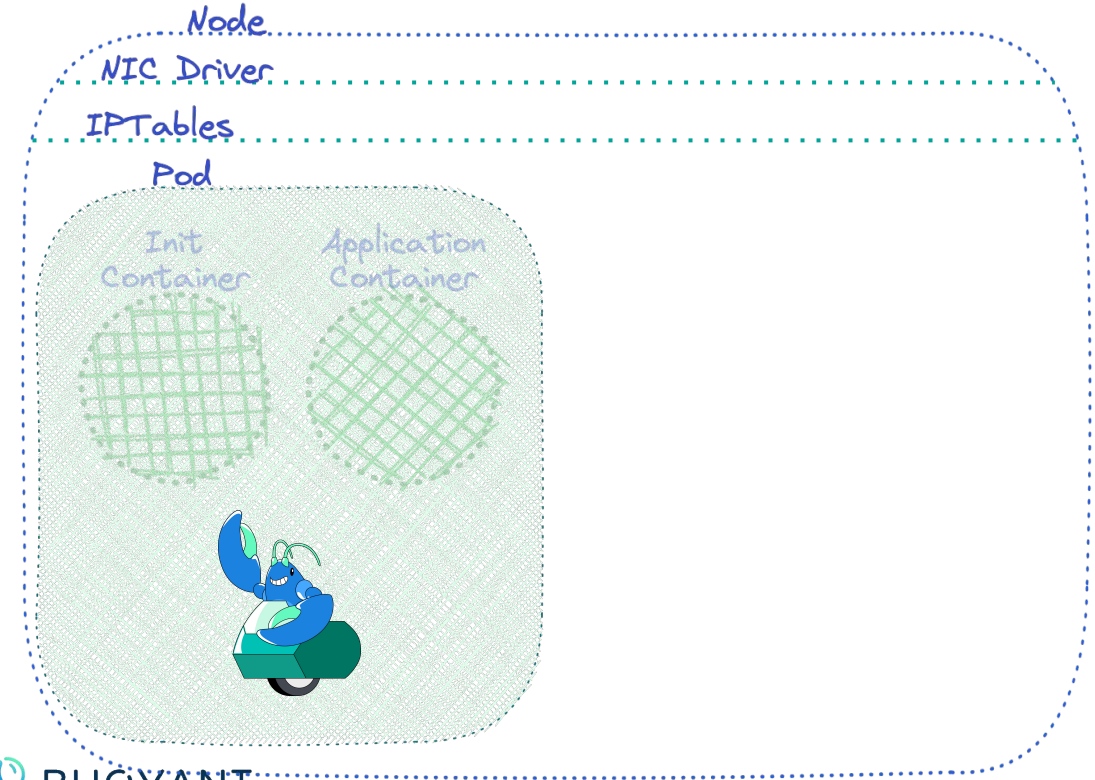
- Pod created with Linkerd init container, Linkerd sidecar, and application container
- Init container starts running
- Init container configures IPTables

Init Container: Happy Path



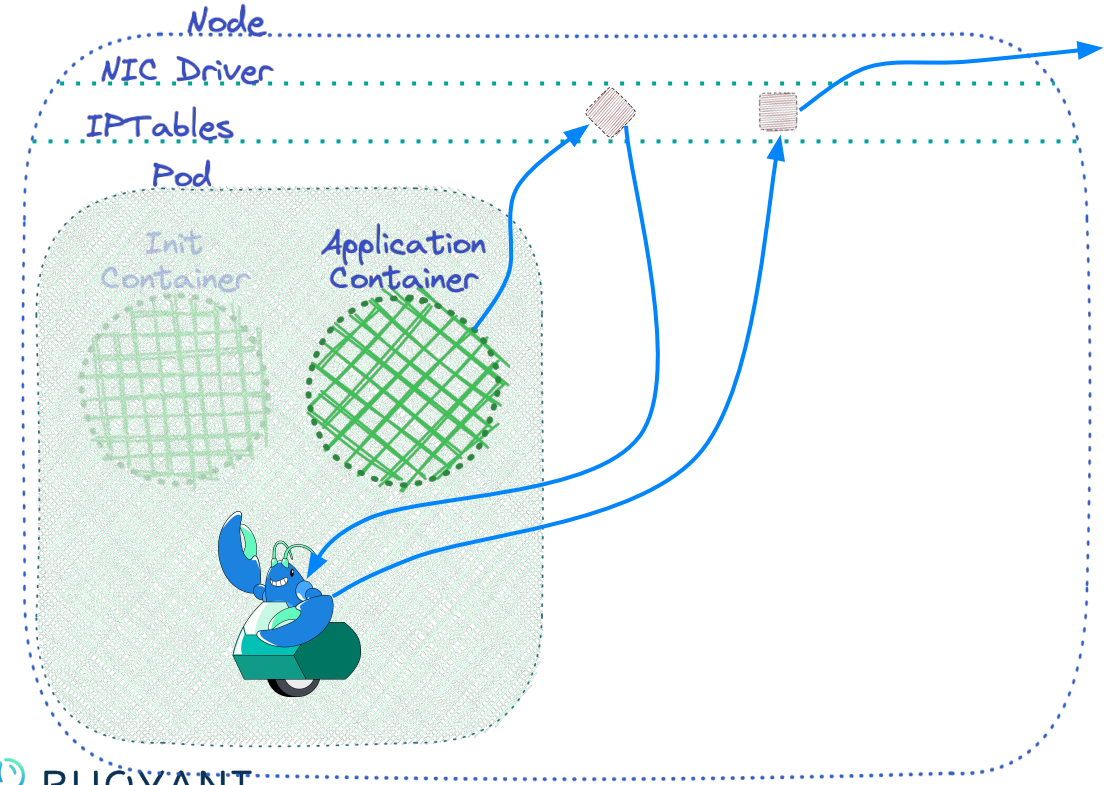
- Init container completes

Init Container: Happy Path



- Linkerd sidecar starts running

Init Container: Happy Path



- Application container starts running
- Application container can make requests through the sidecar

Init Container: less happy paths

- Kubernetes guarantees that the init container finishes before the sidecar starts, but the sidecar is just a normal container.
- Suppose the application pod starts running **before** the Linkerd sidecar?
  - Application won't have access to the network until after the sidecar starts running.
  - Usually self-healing after the sidecar starts, though.
- This is problematic enough that as of Linkerd stable-2.11, we modify the Deployment to prevent it.

Init Container: less happy paths

- As of Linkerd 2.11, `config.linkerd.io/proxy-await=enabled` is the default. With that set, the Linkerd `proxy-injector` will
  - force the Linkerd sidecar to be the first container in the Deployment; and
  - include a `postStart` hook that will wait for the sidecar to be running.
- Kubernetes starts containers in order, and will wait for the `postStart` hook to complete before continuing.
  - Disable this with `config.linkerd.io/proxy-await=disabled`, if needed – that will reintroduce the startup race though!

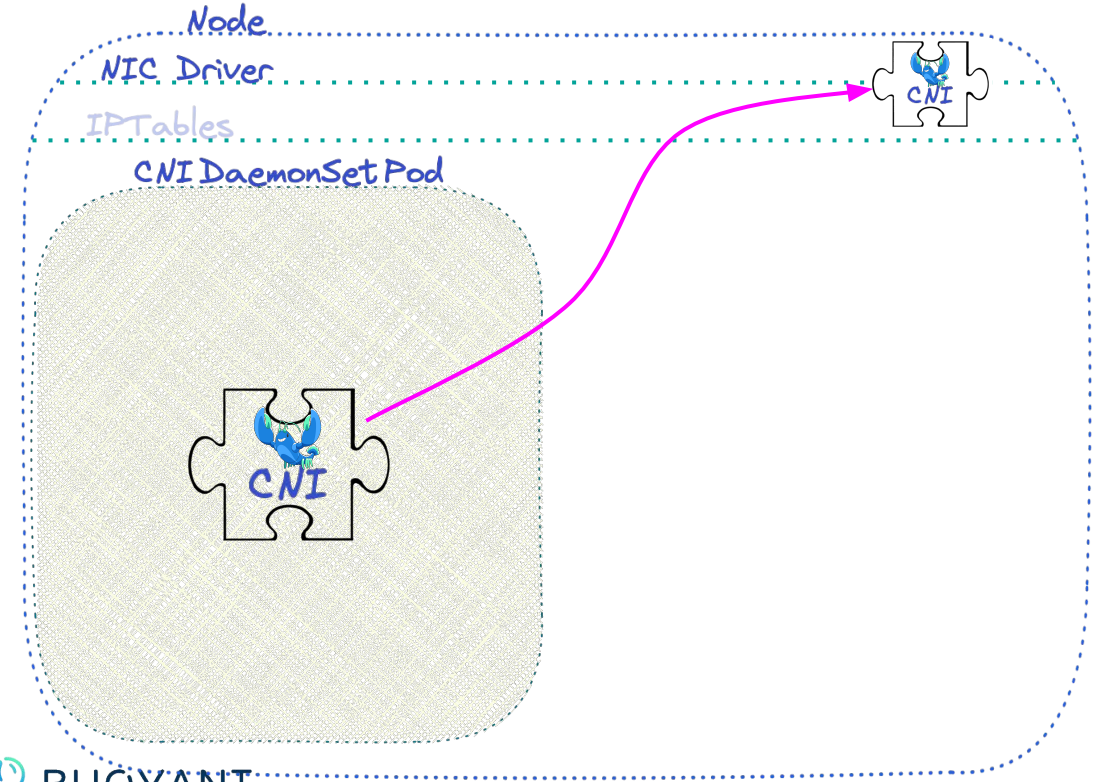
Init Container: less happy paths

- What if the init container doesn't have permission to mess with IPTables??
  - It needs the `NET_ADMIN` capability.
  - Some clusters won't allow this at all.
  - This is a big reason to support CNI plugins!
- What if your OS doesn't support legacy IPTables?
  - This is typical in the Red Hat family.
  - You can `iptables-nft` with `proxyInit.iptablesMode=nft`…
  - …or, again, just use the CNI plugin!

Init Container: typical failure modes

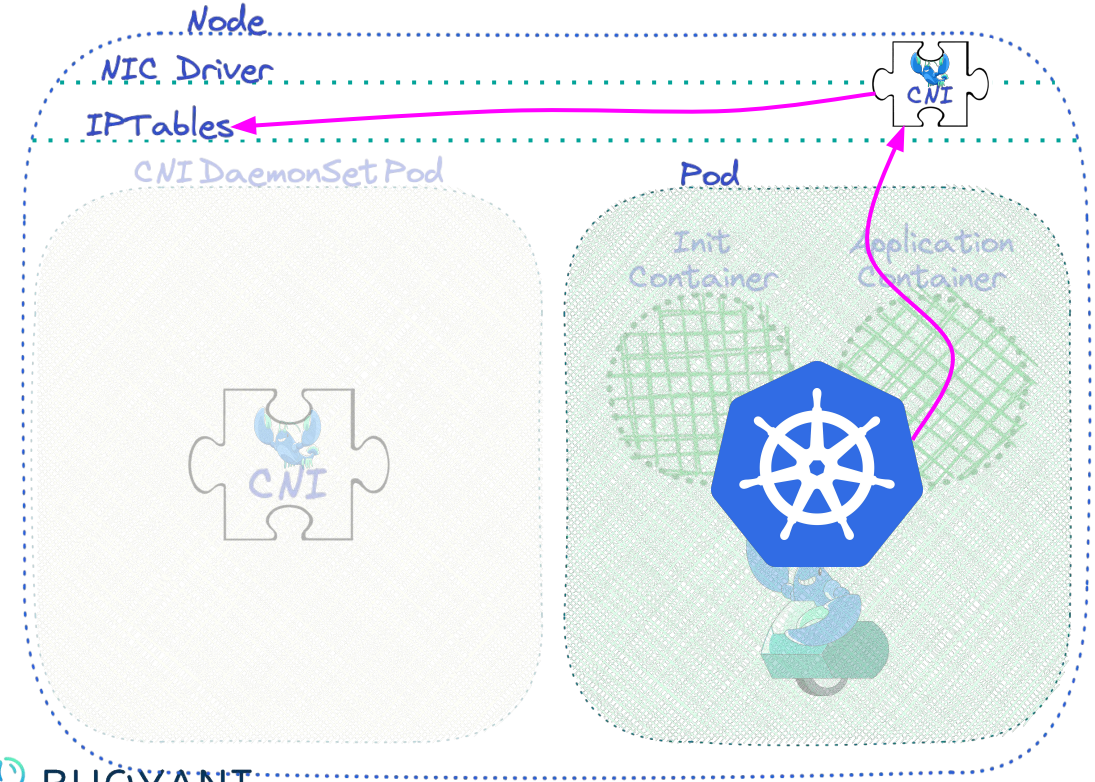
- Most common failure: delays and/or restarts before your Pods are fully running.
  - This implies an ordering issue that's being (possibly slowly) resolved by eventual consistency.
  - `proxy-await` can help a lot here; use it unless you have some really good reason not to!
- If you can't provide `NET_ADMIN`, it's not a subtle failure: your Pods simply won't come up at all.

CNI Plugin: Happy Path



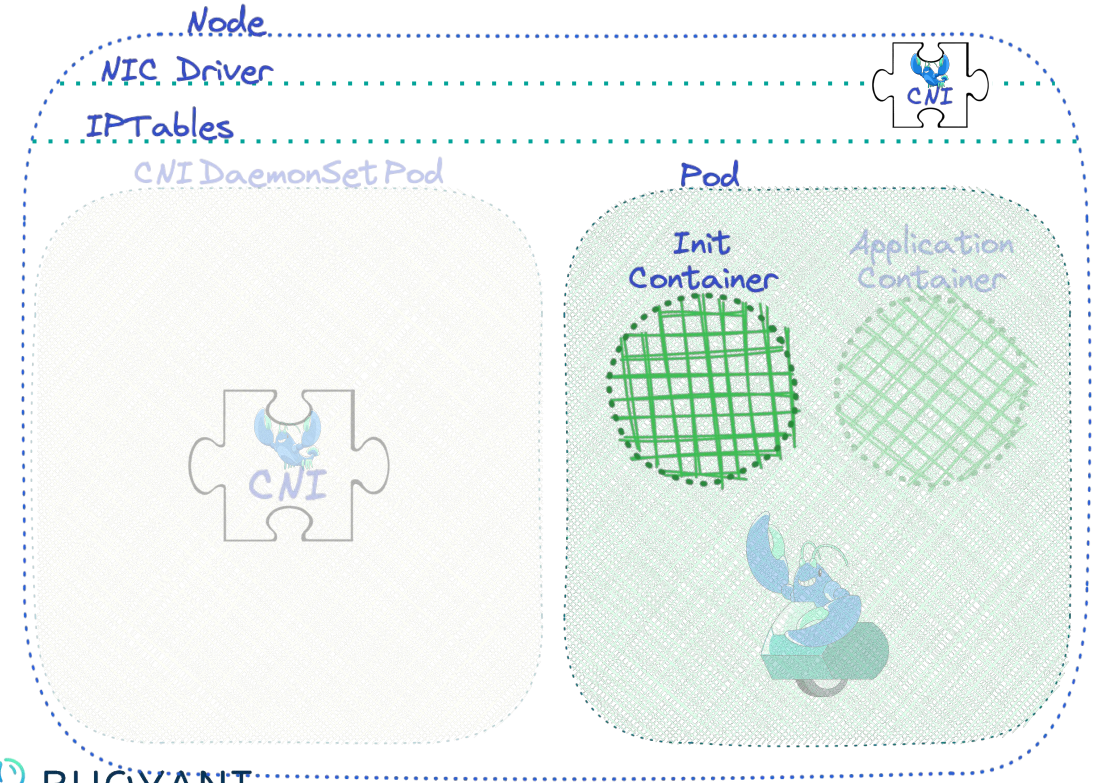
- Node created
- CNI DaemonSet scheduled
- CNI DaemonSet Pod starts running
- CNI DaemonSet Pod installs Linkerd CNI plugin into the Node
- CNI DaemonSet Pod idles, watching for changes to active plugins

CNI Plugin: Happy Path



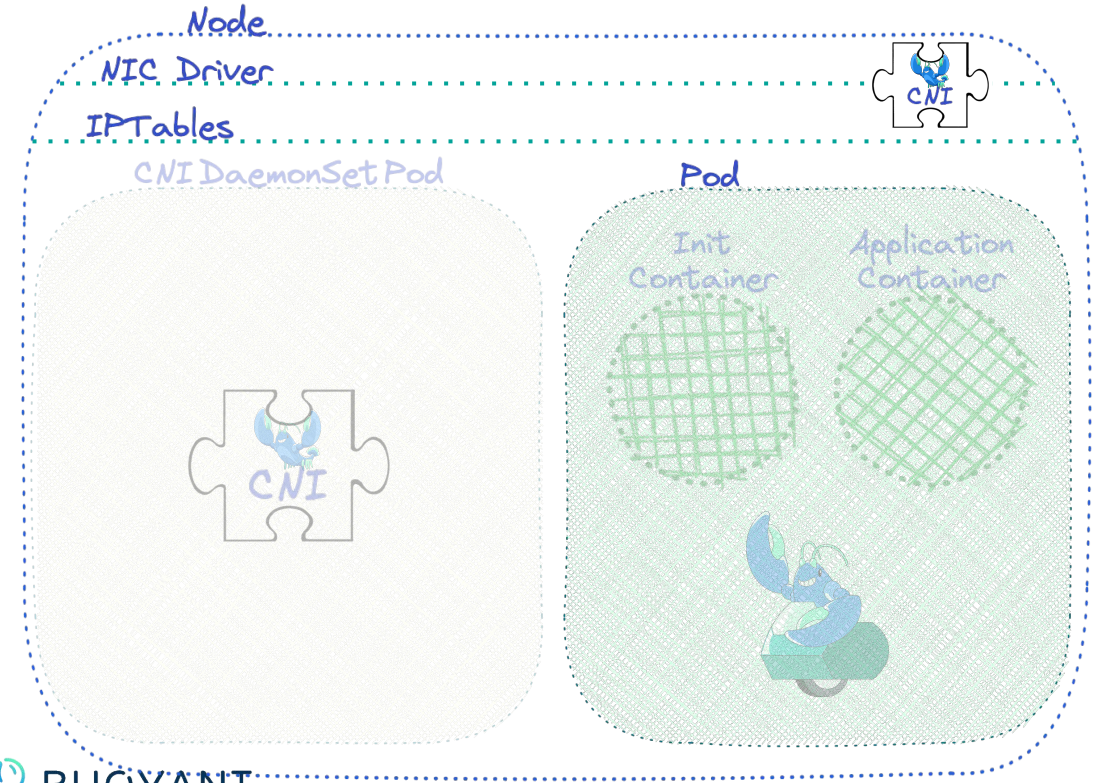
- Pod created
- Container runtime runs CNI plugins (including Linkerd's)
- Linkerd CNI plugin configures IPTables
- Linkerd CNI plugin finishes so that Pod startup can continue

CNI Plugin: Happy Path



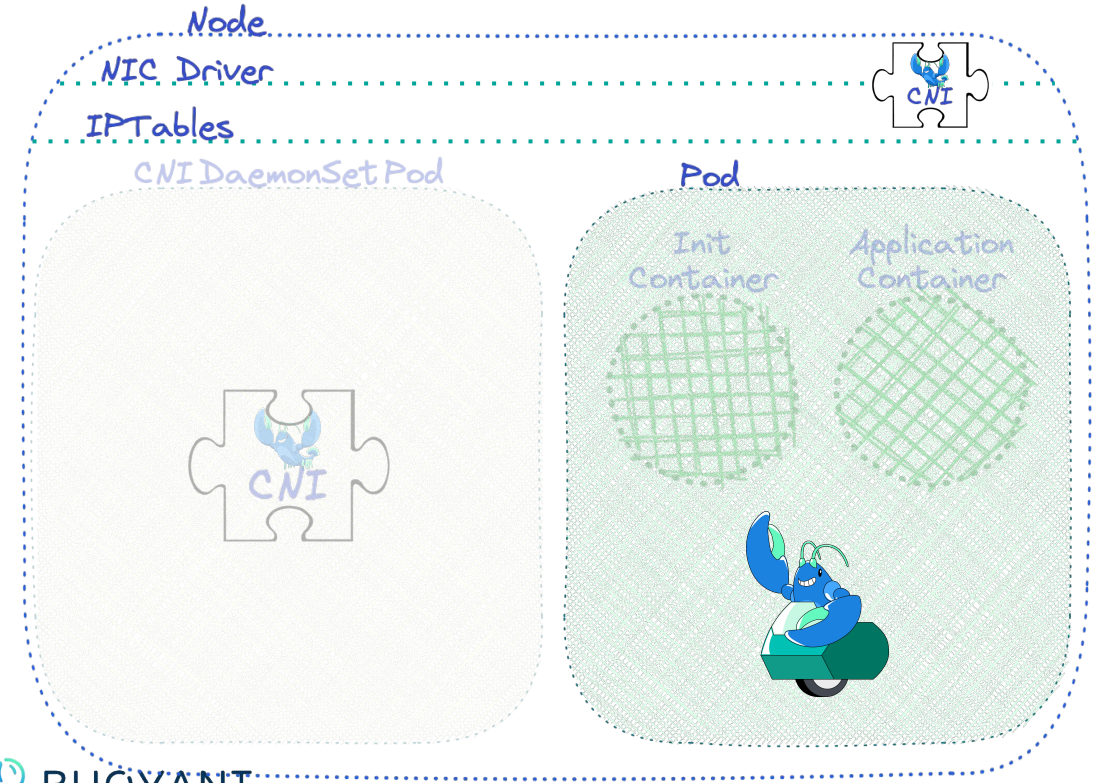
- Init container just validates IPTables setup

CNI Plugin: Happy Path



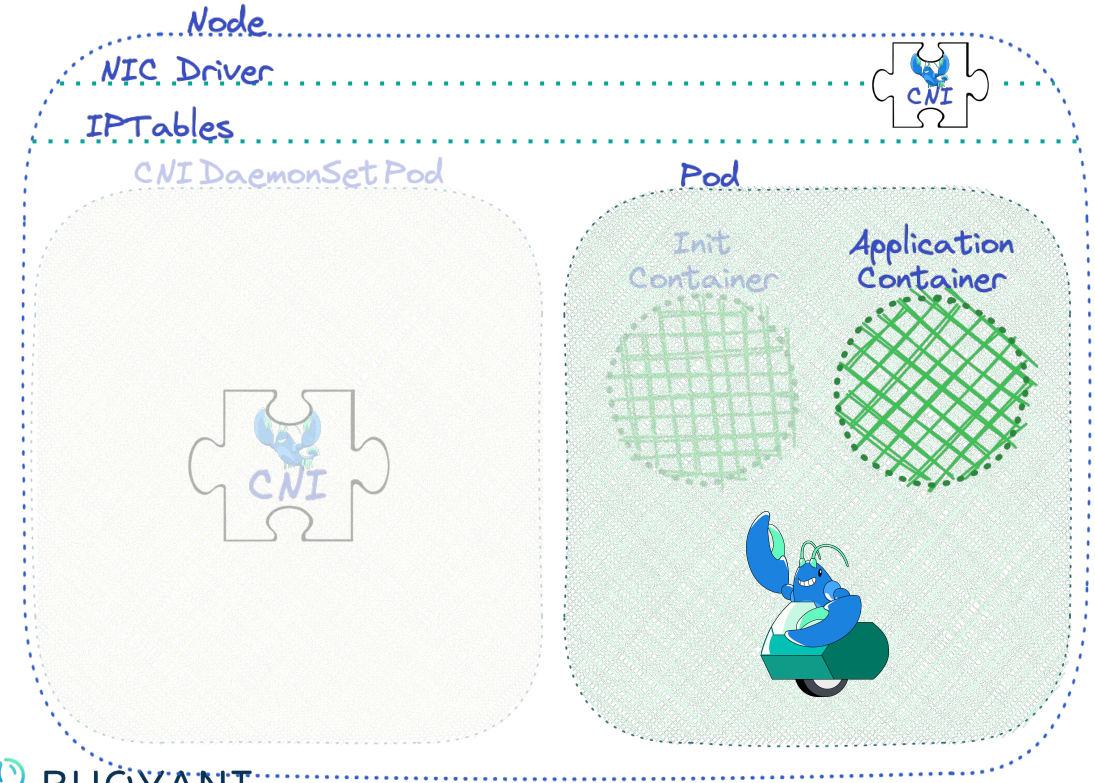
- Init container just validates IPTables setup, then exits.

CNI Plugin: Happy Path



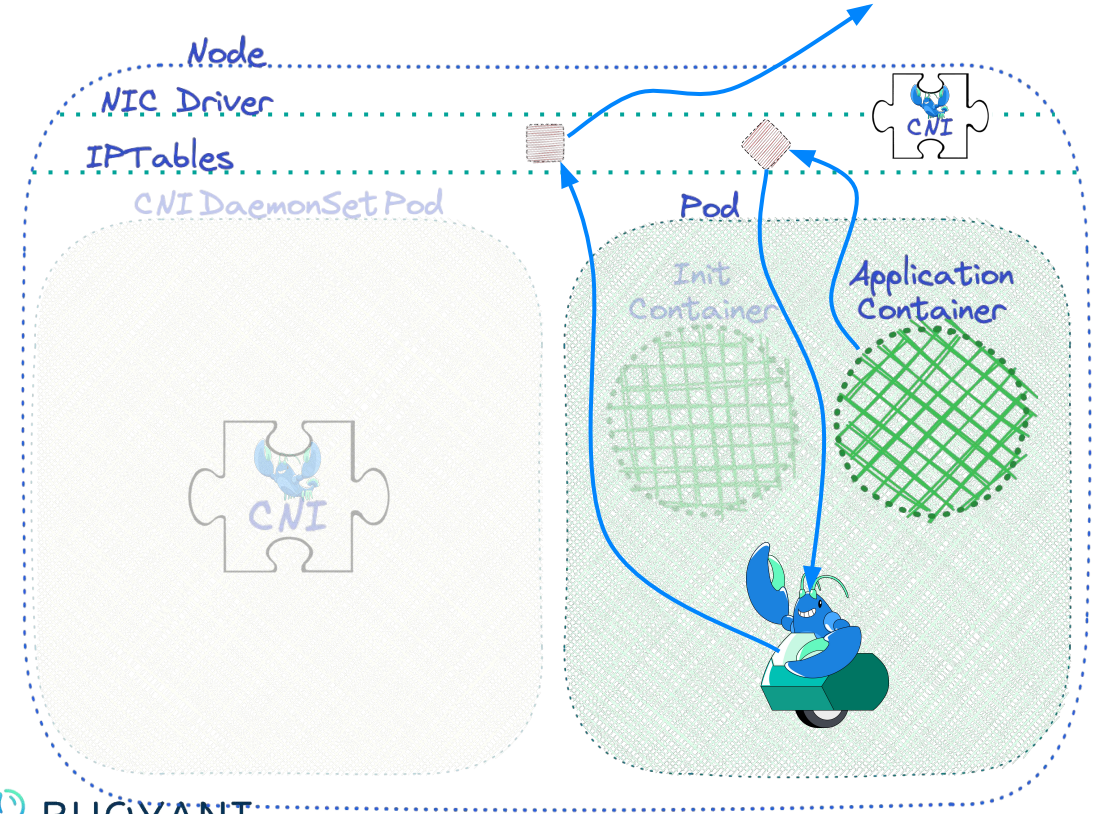
- Init container just validates IPTables setup, then exits.
- Sidecar starts…

CNI Plugin: Happy Path



- Init container just validates IPTables setup, then exits.
- Sidecar starts…
- …then application container.

CNI Plugin: Happy Path



- Init container just validates IPTables setup, then exits
- Sidecar starts…
- …then application container.
- Routing happens as before, with no need for `NET_ADMIN`.

CNI Plugin: less happy paths

- DaemonSet Pods don't get any special treatment during scheduling, so…
- …suppose the application Pod gets scheduled before the CNI DaemonSet Pod is running?
  - Before Linkerd `edge-22.10.3`, nothing will configure IPTables for the application, it will never be able to talk on the network, and the Pod won't start because its health checks will fail. 😕
  - As of `edge-22.10.3`, Linkerd's init container includes a validator to check IPTables and exit if something is wrong
    - This will cause a CrashLoop in the init container, and you'll be able to see that.

CNI Plugin: less happy paths

- Suppose there's more than one CNI plugin in the cluster? (This is actually the common case.)
  - CNI has a way to specify ordering, but it depends on which CNI is in use and is really tailored for cluster providers.
  - Linkerd, instead, uses an installer DaemonSet that watches CNI changes and tries very hard to put the Linkerd CNI plugin last.
  - Again,this will typically prevent Pods from talking on the network, so they won't be able to start running.

CNI Plugin: typical failure modes

- Suppose you misconfigure the CNI plugin in the first place?
  - CNI is a standard, but it's not necessarily installed exactly the same for every kind of cluster.
  - Yet again, Pods will typically not come up because they can't talk on the network at all so their health probes will fail.
  - You may also see corrupt-message errors in the proxy logs for the Pods in the `linkerd` namespace:
    - `{"message": "Failed to connect","error":"received corrupt message"}`

CNI Plugin: typical failure modes

- Good(?) news is that CNI failures are typically not subtle.
- Bad news is that they can all look the same.
- We don't have time to really dig into CNI debugging today, but a couple of notes:
  - Check the installation guides **carefully**.
  - You'll usually need access to the Node to get any real sense of what's up with configuration errors.
  - We're always around at slack.linkerd.io.

BUOYANT

The creators of LINKERD

DEMO TIME 🤞



BUOYANT'S SERVICE MESH ACADEMY

The creators of LINKERD

Monthly hands-on, engineer-focused training from the creators of the service mesh

Dec 15: A deep dive into Kubernetes mTLS with Linkerd



SIGN UP TODAY!

buoyant.io/sma

Fully managed  LINKERD on any Kubernetes cluster

Buoyant Cloud automated upgrades, data plane version tracking, mesh health alerts, and much, much more.

BOOK A DEMO
buoyant.io/demo

BUOYANT

The creators of LINKERD

Thank you!

Flynn

flynn@buoyant.io | @flynn on the Linkerd Slack



@BuoyantIO

buoyant.io