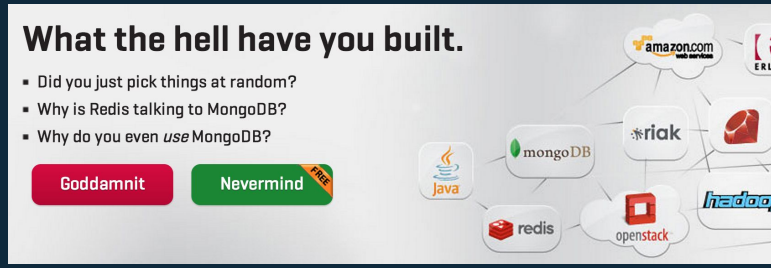
aptomi

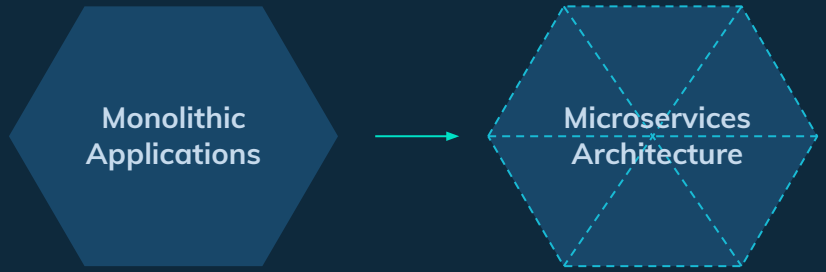
service-centric applications for everyone

Problem



Microservices are taking over the world

⟶ no more monolithic software



Apps decomposed into services...

- ➔ run in containers
- ➔ deployed and scaled separately
- ➔ managed and updated separately

in theory, **but**

... complexity is shifted to a different layer...

... still a hardwired collection of components, without any notion of services

... and it's still hard...

As a result...


Dev *want* to write, push, run *code, updates*

but ***forced to worry*** about ... many **details**

- loads of ops data
- behavioral dependencies
- impact of scaling
- impact of changes/updates
- access control
- security
- location
- networking ····

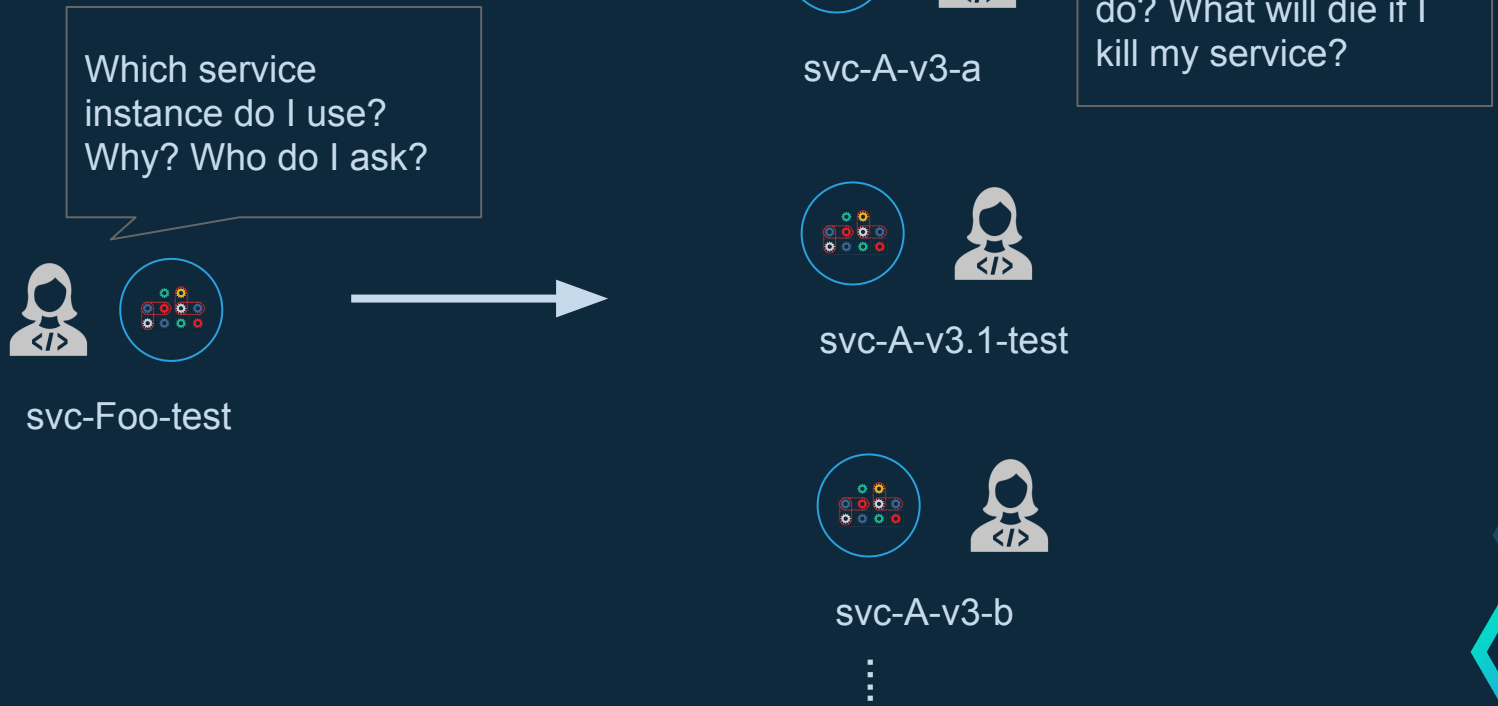
and ***subjected*** to...

- end-to-end **oversight**
- tedious **coordination**
- **synchronous** processes

→ **$$ being spent**

- x Fuzzy **ownership**
- x Lack of **context**
- x Weak **separation** of **concerns**
- x Poor understanding of **impact**
- x Unmanaged **consumption** + **dependencies**

aptomi


Which service instance do I use? Why? Who do I ask?
svc-Foo-test
svc-A-v3-a
Why are they using my stuff? Who's using my stuff? What do I do? What will die if I kill my service?
svc-A-v3.1-test
svc-A-v3-b

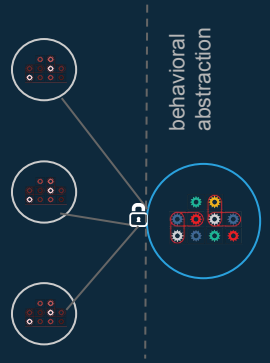
Solution: a missing layer

Monolithic Applications → Microservices Architecture → *Self-driving* applications

Autonomously managed easily composable application services

- ➔ SaaS-like service model for teams
- ➔ Abstraction → reduced complexity
- ➔ Clear ownership → focused expertise
- ➔ Managed consumption → sharing, reuse

Introduce **true service orientation** into microservice management



service oriented abstraction

container orchestration

container run-time

Non-blocking, asynchronous teams
No end-to-end oversight
Operational autonomy

Product

Service-oriented application controller

- ➔ Any app/code → autonomous consumable service
- ➔ Decentralized cross-team/function collaboration
- ➔ Easy updates with contained impact
- ➔ Focus on SLAs and business objectives

offered as a service, used by dev teams

Dev and ops decoupled

SRE robotized

Data →

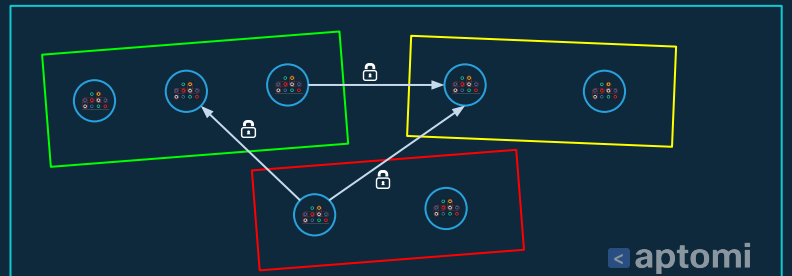
Ops Recipes →

Governance →

service **blueprints** [**slinga** service definition and control language]

define
operate
share
consume

person, machine or CI/CD



Dynamic **Service Exchange**

Authorizer

Realizer

service oriented abstraction

Serverless · PaaS · CaaS · IaaS · Bare Metal

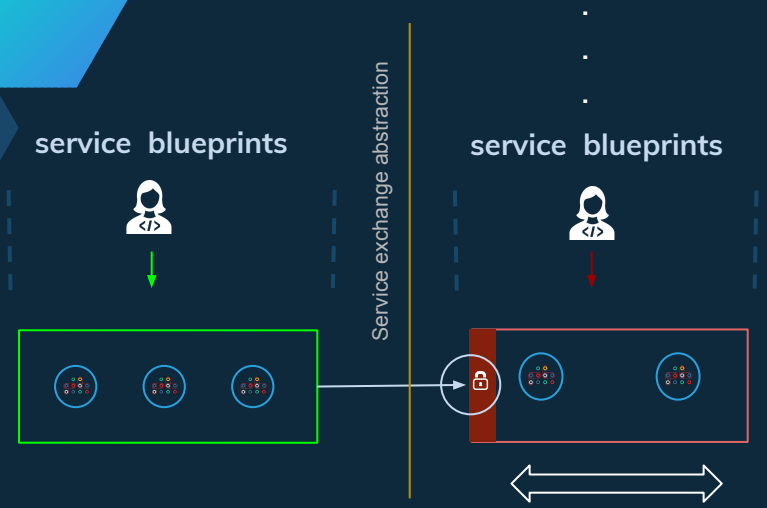
Interchangeable backends and clouds
(initial focus AWS, Kubernetes and Docker)

Public
Amazon EC2, GCE, ...

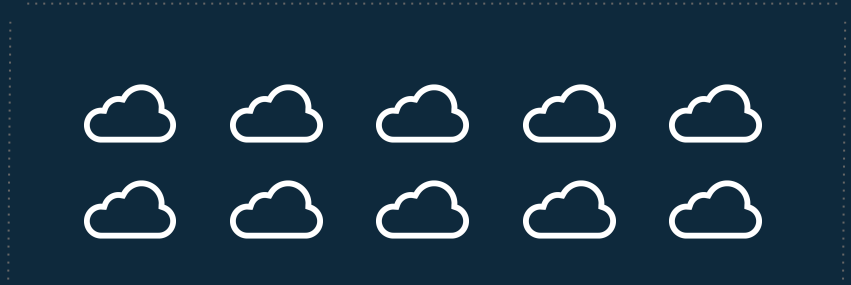
Private
K8s, Mesos, Docker, OpenStack

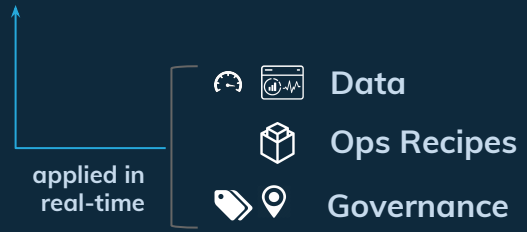
aptomi

Collaborate, but stay in control



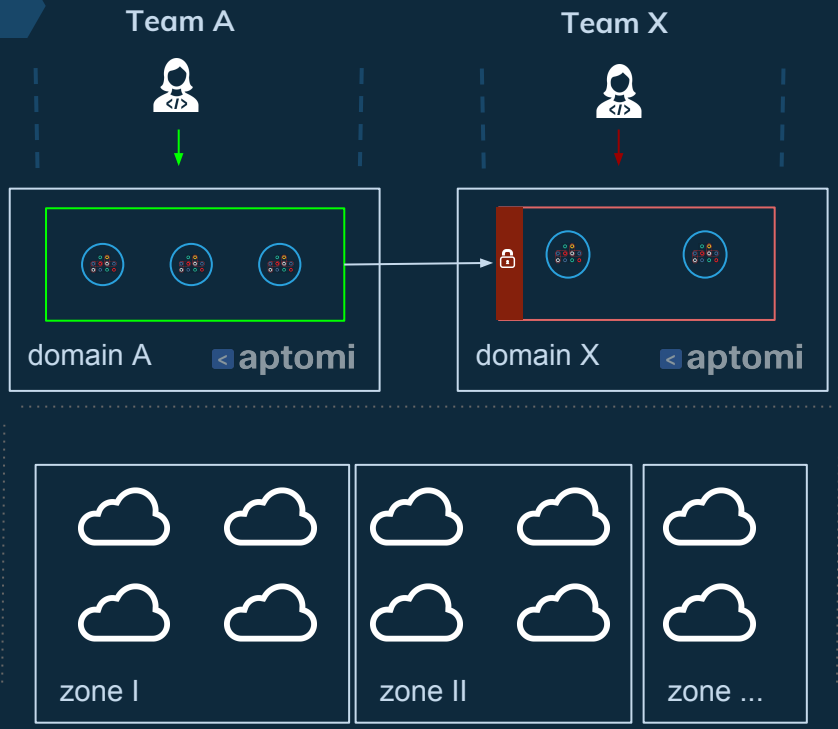
+ Clear ownership boundaries
+ Controlled consumption with authorization
+ Consumer-centric resource allocation and scaling
+ Consumer-centric placement and resource affinity
+ Versioned



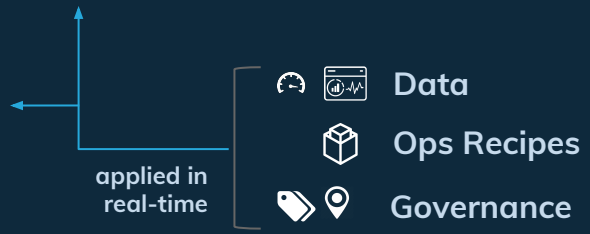


> Taking ops, governance and data into account

Decentralized



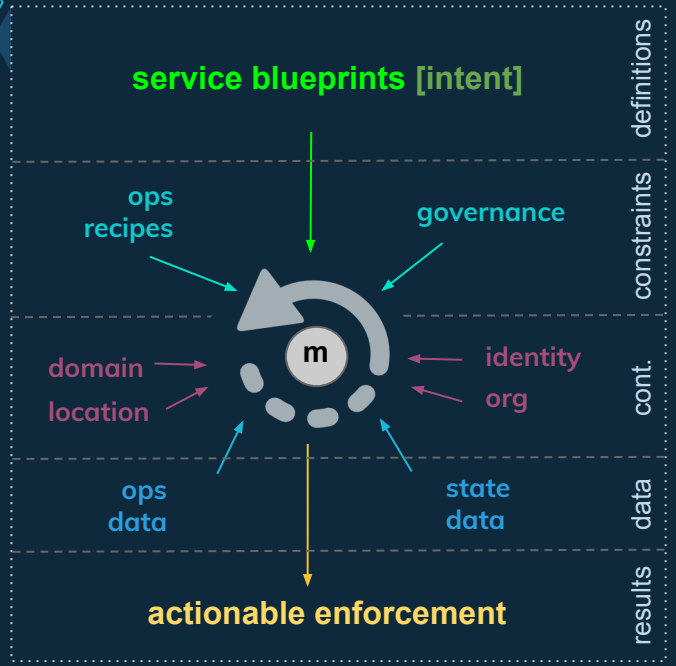
+ Peer-to-peer model
+ No centralized ownership required
+ Domain, cross-domain, global ops + governance
+ Zone specific rules



> Taking ops, governance and data into account

aptomi

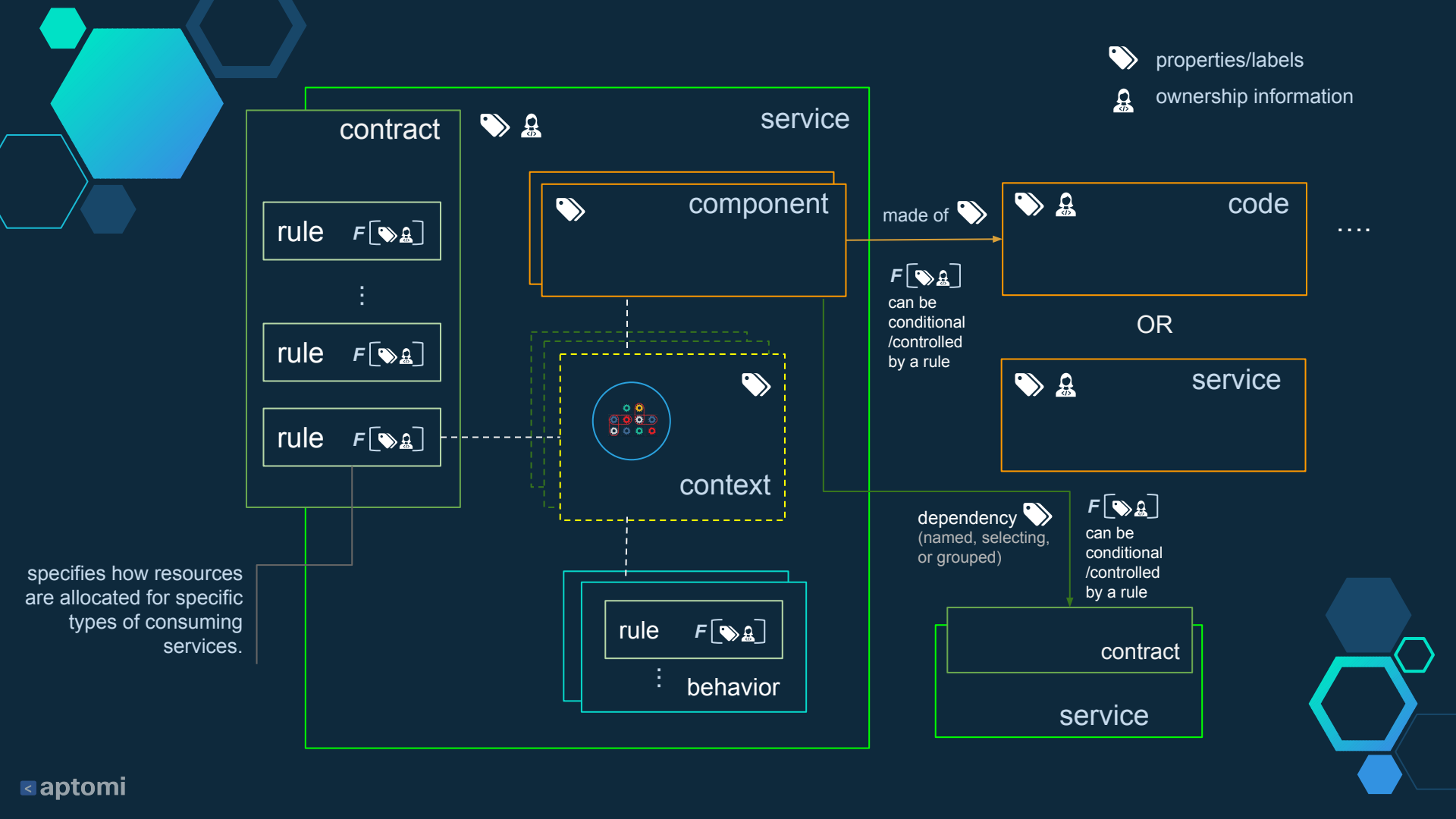
Powered by slinga and match-r

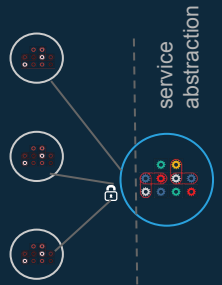


leverage existing knowledge

slinga is a simple but powerful service definition and control language

match-r engine is a distributed framework for real-time matching of service specifications, metadata, ops data, circumstances to actionable rules and enforcement


properties/labels
ownership information
contract
service
rule
F
rule
F
rule
F
component
made of
code
....
F
can be
conditional
/controlled
by a rule
OR
service
context
dependency
(named, selecting,
or grouped)
F
can be
conditional
/controlled
by a rule
specifies how resources
are allocated for specific
types of consuming
services.
rule
F
behavior
contract
service
aptomi



Service Exchange for the service centric enterprise

evolution of the Service Broker

NO aptomi

Service catalogue
(just a pointer to packaged code)

Static templates

Little reuse or sharing of an actual service

Consumer needs to understand how to operate the service

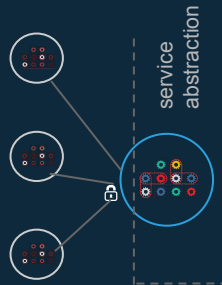
WITH aptomi Service Exchange

Uber for application services.
Automated services controlled by their owners.

Self-contained autonomous services.
Consumable in SaaS-like way.

Highly reusable.
Reuse, share of running services and definitions. Contextualized. No need to keep instantiating more code.

Simplified via abstraction.
Implementation, operational detail abstracted away from the consumer. Service owner is in full control.



slinga service language for the service centric enterprise
→ controllable crowd-sourced application graph

NO aptomi

Static Composition
hard-wired definitions, hard-coded plumbing detail. Suitable for definition of simple components.

Not usable for realistic apps
monolithic composition/landscape gets very complicated very fast

No service, resource dependencies or affinity control. No placements hints.

Additional orchestration/ coordination of order of operations
setup, updates, modifications

WITH aptomi slinga

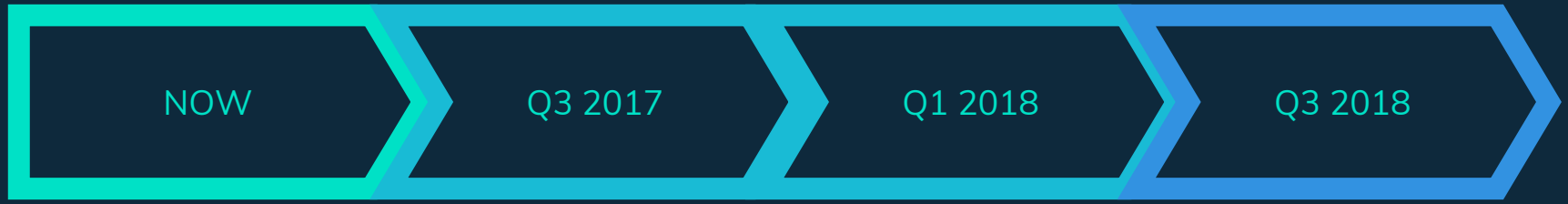
Dynamic Service Definition
Service is a living organism. Self-contained structural and behavioral specification with cross-service and resource dependencies. Ops, governance controls externalized.

Easy to construct complex applications
Modularized composition with abstraction of inner behaviors.

Service dependency, affinity control is central
Intuitive controls of resource allocation per consuming services. Simple resource affinity requirements for allocation of service resources.

Orchestration/coordination built-in
Setup, updates, modifications are coordinated, ordered and orchestrated at service level, with impact minimized.

Plan

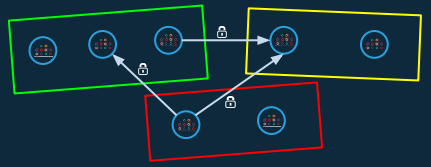


- Open-source project
- Early community engagement

- Evidence of OSS adoption
- 2-3 lighthouse customers (focus on public cloud)
- Launch at KubeCon 2018
- Community outreach

- Enterprise solution MVP
- General Availability
- Partnerships
- Channels

Thank you!



	NO aptomi	WITH aptomi
Setup	**Complex, hardcoded, hard to port.** Too many moving parts. Too many objects to manage. Too much hardwiring.	**Simple, multi cloud portable.** Control what you own. Plumbing and dependencies abstracted.
Workflow	**Confusing and inconsistent.** Static template sprawl, hardcoded dependencies, explicit order dependence	**Clean service model** Concise service oriented abstractions with complete contextualized dependency graph.
Consumption	**Uncontrolled, unversioned.** Static service catalogues offer no rules, sharing or reuse. Consumer exposed to inner complexity.	**Concise, controllable, extensible, versioned.** Dynamic services with inner operational detail hidden by the service abstraction. Controllable by owner via understanding of “consumer’s needs.” SLAs.
Management	**Context and understanding of impact missing, poor versioning.** Uncontrolled dependencies and understanding of potential effect of actions.	**Managed versions and stacks. Full consideration of dependencies and context.** Full dependency graph, clear understanding of impact and context.

Business Value

Cut Operational Costs
Enable development teams to do more rapid iterations.

Reduce Time to Market
Empower Business Units to focus on services, not plumbing.

Increase Reliability
Make service deployment and updating predictable.

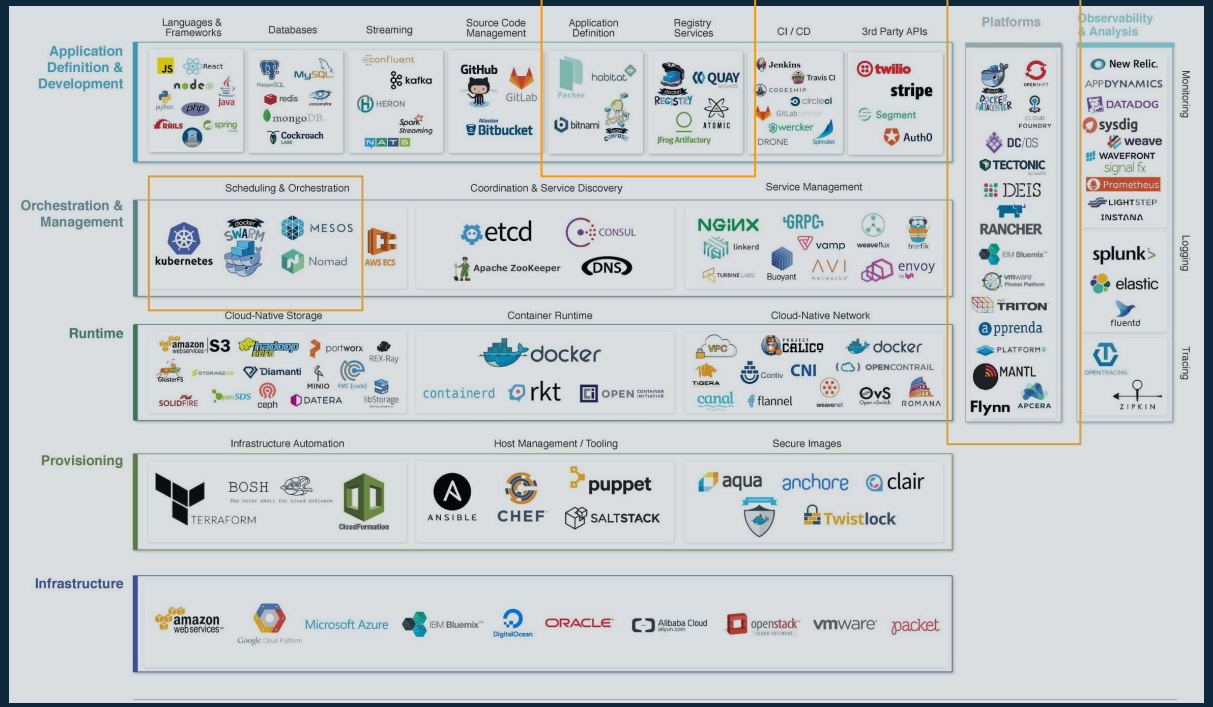
Minimize Downtime
Remove unforeseen errors from unknown dependencies.

Liberate Dev Cycles
Decentralized service ownership. Higher re-use.

Improve Insight
Greater contextual visibility and introspection.

Competitive Landscape

Lots of activity, no focus on services and team collaboration



PaaS is outdated

Containers by themselves don't solve the problem

Container orchestration focuses on collections of containers and not relationships

→ disruption is coming

Insertion, GTM, ...

LoB Developer		Head of LoB Dev/Ops, Platforms or CIO	
Service Producer	Service Consumer	Service Producer	Service Consumer
Easily define and operate services. Don't worry about plumbing.	Consume services without understanding of inner workings.	Advanced contextual visibility. Secure multi-tenancy.	Integrated security. Integrated auth/authz (IAM, AD) Auditing Team Collab
Open Source Github CNCF Kubernetes SIGs		**Enterprise Solution** SaaS Subscription Model Support AWS/Azure/GCP	

→ inside sales + organic growth within broader org
→ no central ownership required

Focus on cloud/container/microservice initiatives, new projects, major updates, migrations, tool-chain refresh