

A blurred background image showing two people in an office. On the left, a person with dark hair is wearing a dark red sweater and pointing their right index finger towards a computer monitor. On the right, a person with light brown hair is seen from the side, looking at the same monitor. The office environment is out of focus, with other monitors and office equipment visible in the background.

Using a managed container service for running and scaling Kubernetes applications in the cloud

Murray Smart

GM Tech Delivery - Canstar

Developing mature DevOps practices to operationalise large scale deployments

DevOps practices for large scale deployments

- Communication

- Collaborating Teams

- Context

- Customer Satisfaction

- Process

- Continuous Integration (CI)
- Continuous Deployment (CD)

- Tooling

- Effective Tools
- Test Automation

Communication

- Collaborating Teams
 - Trust each other
 - Understand the value of role distinction
 - Leverage the strengths of team members
 - Self organise
 - Own what they build
 - Always willing to grow and learn new skills

Context

- Customer Satisfaction
 - Purpose built solutions
 - Monitoring for success
 - Responsive feedback loop

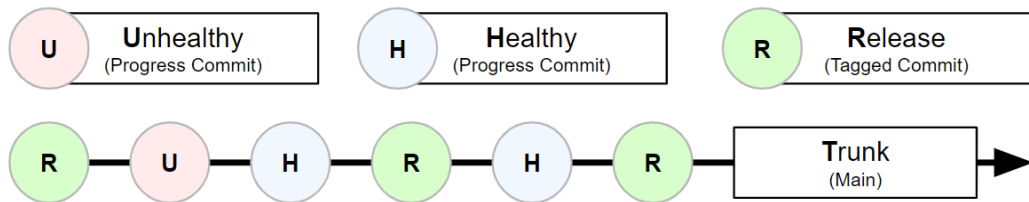
Process

- Continuous Integration (CI)

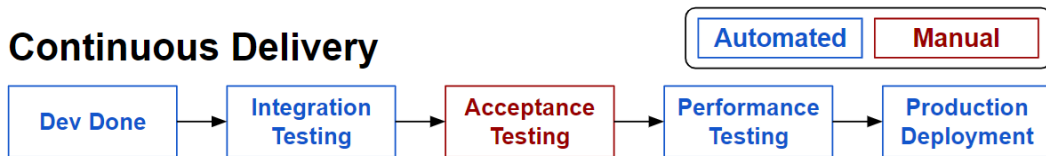
- Trunk based development
- Feature flags (build or deploy stage)
- Mob programming
- Established coding standards

- Continuous Deployment (CD)

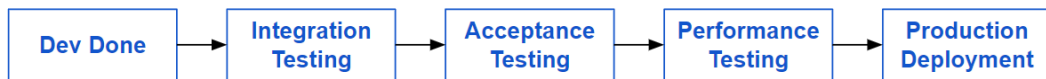
- Requires automated testing
- Continuous delivery (stepping stone)



Continuous Delivery



Continuous Deployment



Tooling

- Effective Tools

- Version Control
 - Infrastructure as Code (Hardware)
 - Configuration as Code (Software)
 - Source Code (Application)
- Pipeline Management
- Automation
- Documentation

- Test Automation

- Input vs Outcome not Implementation
- Confidence to Release is the Goal



Combining Kubernetes and cloud native managed services to dynamically scale

Why Kubernetes?

- Container orchestration that scales
- Configurable auto-pod migration on node failures
- Easy debugging / pod management
- Network configuration

- Easy resource management
- Deployment continuity
- Multi-cloud capable
- Rolling update support
- Strong cloud native tool integration

Why Cloud Native Managed Services?

- **Multiple support options**
 - Online communities
 - Documentation and guides
 - Often includes direct provider support
- **Dynamic scalability**
 - Rapid access to additional infrastructure and resources on demand
- **No infrastructure maintenance costs**
- **Disaster recovery**
- **Centralised control**
 - One stop shop for entire cloud computing stacks
 - Logging
- **Provider migration options**
- **Services designed for low effort integration**

Putting it all together

- **Infrastructure as Code**

- Reduce DR impact potential
- Faster debugging
- Consistency and reusability
- Reduced costs by shutting down internally used node groups outside business hours (i.e. UAT)

- **Debugging**

- Multiple debug / logging options to suit

- **Provider agnostic**

- Cloud native managed service provides all support K8s

- **Multicloud K8s clusters**

- Centralise dependent containers from multiple providers
- Reduce latency and related performance constraints

- **Configuration as Code**

- Dynamically scale environments that have a mix of shared and dedicated services

- **Spot Instances / VMs**

- Save costs on fault-tolerant workloads



Increasing rapid release cycles and flexibility in cloud environments

Foundations of rapid, flexible release cycles in cloud environments

- Leverage CI/CD services

- Automate testing
- Automate builds
- Automate deployments
- Automate everything!

- Environment consistency

- Scale / Resources
- Configuration
- Code

- Simplicity

- Use multi-tenancy where colocation makes sense
- Smaller release scope

- People

- Documentation
- Monitoring and alerts
- Self organising teams

Leverage CI/CD services

- Automate testing
 - Contract vs End-to-End testing
 - Regression testing
- Automate deployments
 - Automatic rollbacks
 - Rolling deployments
- Automate builds
 - Git lifecycle integration
 - Atomic build artifacts
- Automate everything!
 - If you can describe it as a repeatable process, it can probably be automated

Environment consistency

- Scale / Resources

- HorizontalPodAutoscaler (HPA) can reduce scale while maintaining consistent resource allocation per pod across environments for swift bug replication and resolution

- Configuration

- CaC eliminates the potential for environments to fail due to unique configuration settings
 - Faster debugging process
 - Fix once, apply everywhere

- Code

- Leverage a consistent branch naming convention to help track atomic build artifacts
- Use pipelines to catch issues that can impact release flow
 - i.e. Cyclomatic complexity, security vulnerabilities, etc.

Simplicity

- Use multi-tenancy where colocation makes sense
 - Reduce unnecessary service maintenance
 - Save on unnecessary infrastructure costs
 - Less effort to scale when needed
- Smaller release scope
 - Lean into feature flags
 - Prefer no-op modules over conditional statements
 - Reduce cross service dependencies in a single release where possible

People

- Documentation

- SemVer
- Automated change logs driven by commit messaging
- Troubleshooting guides help reduce delays when common, low priority issues are encountered

- Monitoring and alerts

- Early detection of failures
- Proactive monitoring allows for preventative changes to be implemented via IaC or CaC keeping pressure off the release flow

- Self organising teams

- Understand the nature of changes being released
 - Reduces downtime when merge conflicts are encountered
- Own the environments depended on for rapid release cycles



Addressing security, networking, complexity and monitoring challenges

Security Challenges

1. Pods without NetworkPolicy coverage have zero network restrictions resulting in reduced overall security
2. Kubernetes out of the box security solutions are not comprehensive
3. Using NodePort increases your attack surface

Security Recommendations

1. Restrict pod communications by writing sensible NetworkPolicies and using Role-Based Access Control (RBAC)
2. Third party tools and cloud native managed service providers provide substantial security cover
3. Switching to Ingress is an effective way to secure the cluster with a single point of entry

Complexity Challenges

1. Unexpected behaviour within a Kubernetes cluster can be difficult to track down and fix due to how many moving pieces are involved under the hood
2. Breaking down monolith services into single purpose microservices while they are still in active use

Complexity Recommendations

1. Leverage software / services that perform the diagnostic heavy tasks (often powered by AI)
2. Isolate distinct functional requirements of the monolith into separate microservices
 - a. Put each microservice on a pod in a Kubernetes cluster to make it easy to scale and implement service discovery

Networking Challenges

1. Network configuration in large scale Kubernetes deployments is complicated
2. Network security configuration complexity can become a bottleneck when onboarding new team members or debugging issues

Networking Recommendations

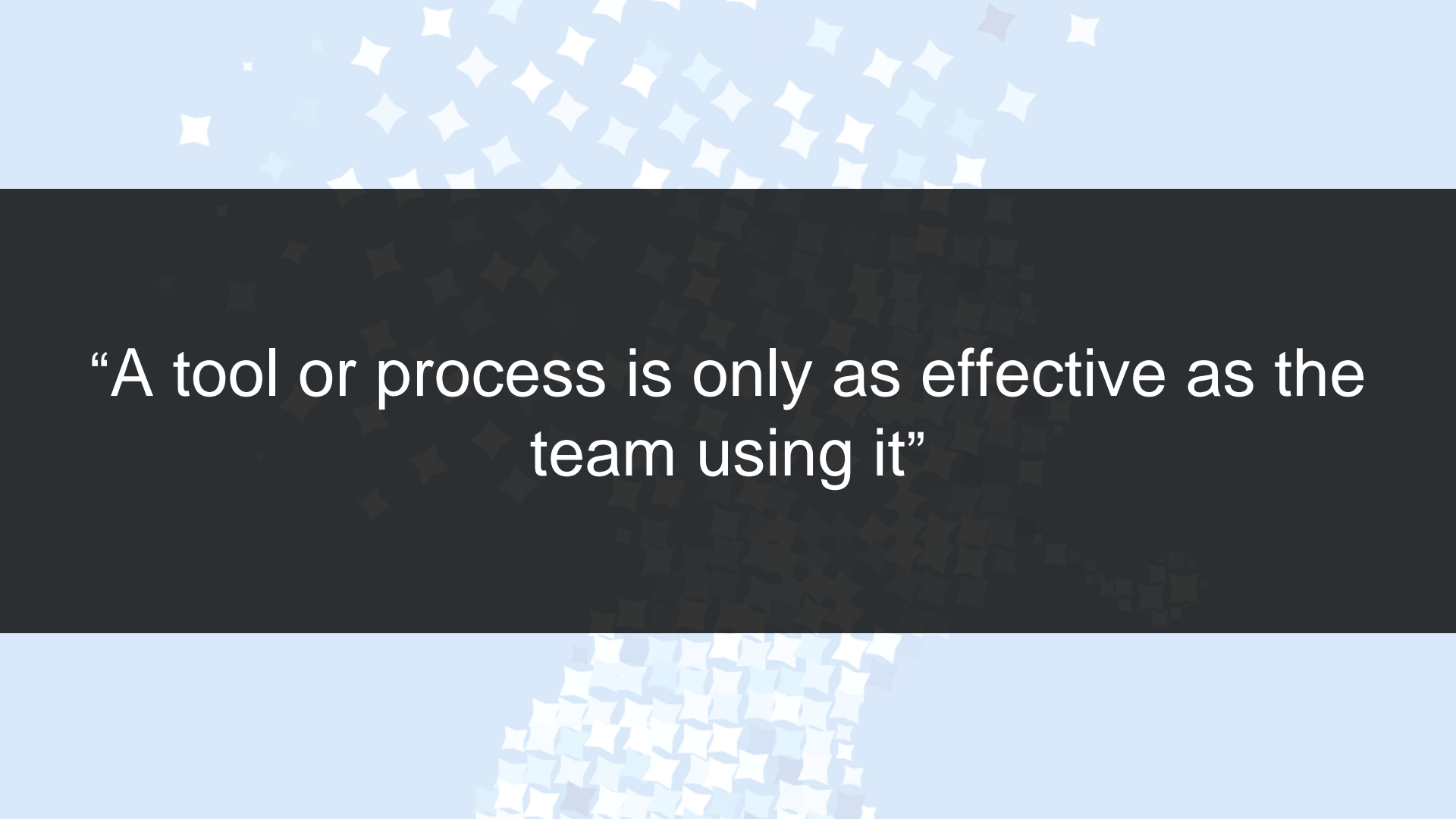
1. Consider the Pros and Cons of available Container Network Interfaces (CNIs), and choose one or more that address deployment requirements with simplicity as a primary success criteria
2. Clear documentation can help speed up onboarding and debugging processes

Monitoring Challenges

1. An emphasis on monitoring via health check metrics can result in critical issues being overlooked
2. At scale it can become a time consuming task to sort through the sheer volume of observability data that is collected

Monitoring Recommendations

1. Look for solutions that are capable of leveraging multiple signal types to properly observe the status of services and infrastructure
2. Use services that can do the heavy lifting on processing collected observability data so teams can focus on diagnostics / issue resolution



“A tool or process is only as effective as the
team using it”

Thank You