Don’t Get Caught in a Blizzard!
Avoiding Snowflakes in Cloud Migrations
Chaos Monkeys love “snowflakes” — those one-off, special services and setups that are particularly vulnerable to crashes and problems with application and infrastructure management. This talk gives a behind-the-scenes look at how HUIT’s Cloud & DevOps group avoided snowflakes when migrating to the cloud while also making it faster and easier for service teams to get their work done.

• Introduction (5 min)
• Why Are We Talking About Snowflakes? (5 min)
• Snowflakes (10 min)
• Blizzard in the Cloud (5 min)
• Patterns and Standard Components Calm the Storm (15 min)
• Questions (10 min)
Who Are We?

• Joel Fanton
  Director of DevOps, HUIT

• Magnus Bjorkman
  Director of Solution Architecture, HUIT
Why Are We Talking About Snowflakes?

Introducing the Cloud Migration Project.

• Per mandate from Harvard CIO Anne Margulies, we will migrate approximately 75% of 600 existing applications — and all new applications — to the cloud

• The Cloud Migration Team is taking a DevOps approach to migration, and is currently building an automation engine to provide for:
  • rapid, consistent cloud infrastructure deployment
  • continuous application integration, testing, and delivery
  • built-in fault tolerance and security best practices
  • highly available resources and disaster recovery

• As part of our DevOps approach to cloud development, infrastructure and resource creation should be repeatable and predictable

• Once apps have been migrated, operational support must be sustainable

• “Resource snowflakes” are a major deterrent to repeatability, predictability, and operational sustainability
What Are Snowflakes?

- Servers uniquely configured for specific projects and applications, often using nonstandard tools and processes
- Systems that suffer config drift and document rot over time
- Automation can help rectify some problems, but you can still have snowflakes with automation
Snowflakes: Setting Up New Systems

Setting up new systems poses a special set of challenges:

• Unique systems take a longer time to set up (often weeks) — usually by multiple people specializing in distinct technologies

• **Testing** that components work together as a whole takes additional time, which lengthens the overall timeline

• Must have specialists available for components that need to be configured; hand-off and scheduling will take additional time
Snowflakes: Maintaining Existing Systems

There are problems with maintaining existing systems, too:

• If you have 1,000 unique setups, you’ll need to do every update in 1,000 different ways — **this is the larger and accumulating cost**

• How do you update for a zero-day attack across 1,000 unique setups in a quick, controlled way? — it requires a lot of people

• How do you integrate new tools across 1,000 unique setups? — with local resources, over a very long time
Snowflakes: When Something Bad Happens

What happens when a “snowflake” system crashes?

• How do you quickly recreate it?
  • Find people that have knowledge about it
  • Find current documentation
  • How far does the backup get us?

• Do you mirror the production environment for testing?

• Do you create another instance to add to a cluster, even if you can’t guarantee exactly the same setup?
Managing a farm of snowflakes poses some problems:

• How do you know what to monitor? (manual setup and manual track)
• How do you identify what each server does? (manually document)
• How do you detect what works and what doesn’t?
• How do you know that you follow best practices, including security?
Here Comes the Blizzard!

What happens when we move to the cloud?

• It’s even easier to create snowflakes — everything you need is at your fingertips
• It’s fast and easy to create new systems ... so you can get a lot of systems
• Systems can be set up automatically depending on your traffic and needs
Blizzard: Losing Track of Systems

Over time, it becomes very easy to lose track of the systems you have.

- **Orphaned instances:** How do you keep track of which servers are actually running important workloads? It can become Russian roulette if you don’t have a standard and automated way to track usage.

- **Security holes:** How do you detect config drift and the risk of data theft?

- **Unexpected costs:** If you don’t manage the scale of your environments (especially non-prod), you’ll very likely have higher costs than on-premise.

Port: 80
Add Port: 3389
Blizzard: Keeping Tabs on What You Have

How do you keep control of rapidly proliferating instances?

• Use standardized methods for monitoring and tracking instances so you can determine what to run and what to terminate
• Use standardized ways to detect config drift and security holes
• Only run instances/environments when needed — otherwise, destroy them
The Need: Continuous Setup and Teardown

Run it when you need it!

• Tear it down when we don’t need it
• Set it up again when we need it

As we keep rebuilding, we need to handle changes to standards and components:

• A setup process that accounts for change
• A process to test over time that changes are not disruptive or detrimental
How Do We Do This?

• Since everything is infrastructure-as-code, components can be developed once, tested, and then used many times.

• Build a component once, build it well, and make it reusable across the board.

• Make components independent but tested to work together — this enables users to choose which ones they want to use.

• Make components integrate with standardized tools to offer visibility and management of the farm.

• Make tools self-service and integrate with cloud providers, so users can take advantage of the cloud’s speed and control.
Patterns & Standard Components: New Systems

Key points when it comes to setting up new systems:

• Quick setup using automated self-service components (can be minutes!)
• Choose from a selection of tested components needed for applications
• Standardized ways to set application-specific configurations
• Specialist knowledge is baked into components, reducing the level of expertise required to set up infrastructure
What about maintaining existing systems?

• Update in one place, then roll out to every user

• In the case of a zero-day attack, test updates in lower environment and then roll out in production

• When introducing new tools, one place to swap out tools and test against component contract
What about solutions when something bad happens?

- **A system crashes?** Your automated setup of standardized components can be run at any time by anyone

- **Mirror production environment for testing?** Keep running the environment setup as many times as you want

- **Create another instance to add to a cluster?** This happens automatically as part of integrating with cloud providers and automated setup using standardized components
Patterns & Standard Components: Managing the Farm

• How do you know what to monitor? Standardized components are automatically discovered and tagged with information that determines usage

• How do you identify what each server does? Standardized tagging will sync with information in ServiceNow

• How do you detect what works and what doesn’t? Standardized endpoints and configurations allow automatic discovery of what’s healthy and what’s not

• How do you know you follow best practices, including security? Standardized practices and policies can be codified and compared with the discoverability of standardized components
Additional Best Practices

• Using a micro services architecture to manage and deliver patterns and components for frequent releases and scalability

• Continuous integration and regression testing of all patterns/components

• Versioning of interfaces and features for backward compatibility and stability

• Documented, predictable extension points
Current Activities

• Building the micro service infrastructure and model
• Building an initial set of patterns and components, with a focus on the LAMP stack
• Preparing to work with application teams to incorporate use cases and requirements
• Working with the HUIT Software Standardization/Software Engineering Working Group to develop a guide for migrating applications to the cloud
• Starting work with teams with existing cloud deployments
Questions?
Thank you!