solarwinds

Innovate Summit 2017

Infrastructure as Code

Fredrik Skogman

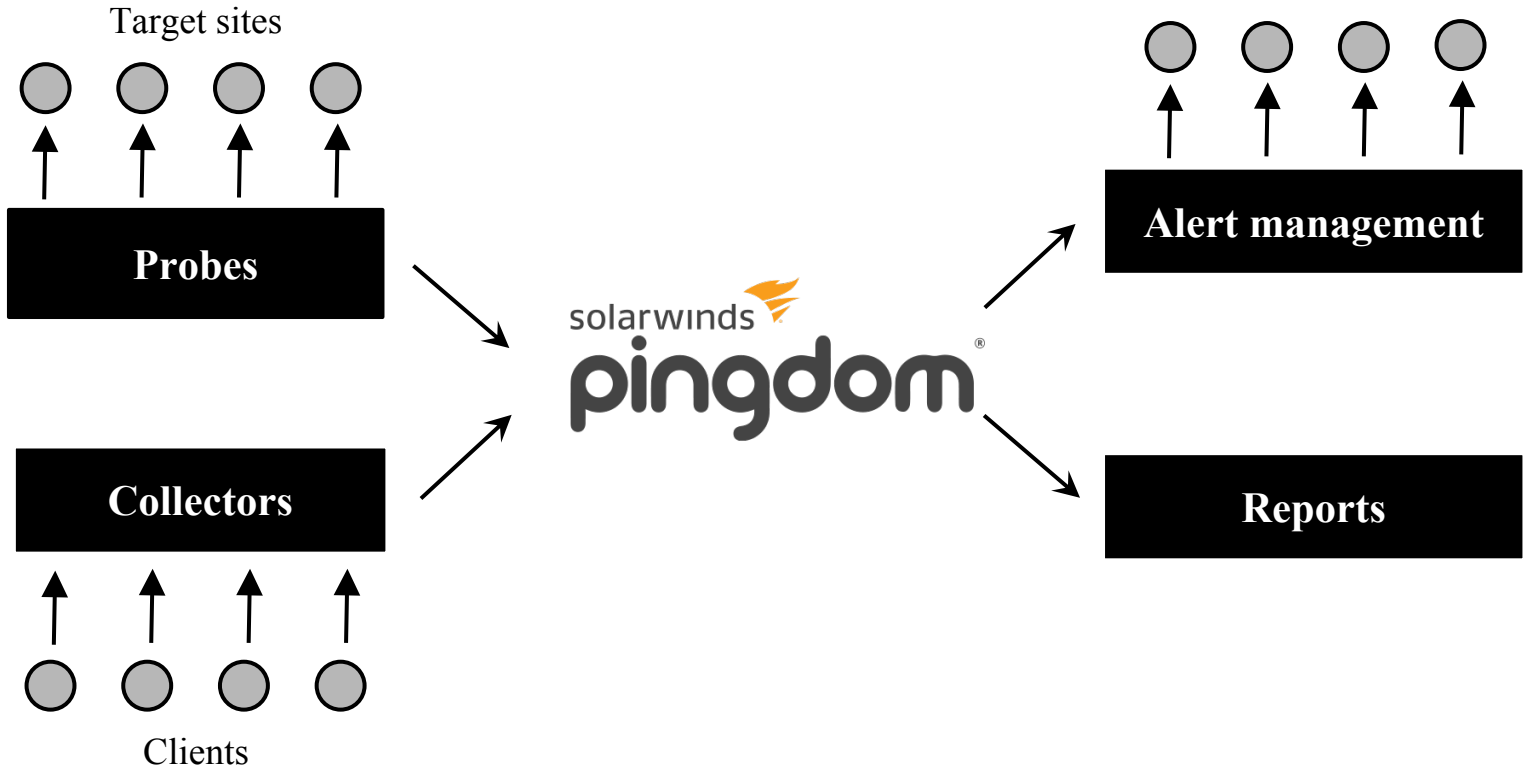
Hi, I'm Fredrik

solarwinds

Fredrik Skogman

- Principal Architect at Pingdom, part of SolarWinds Cloud
- Located in Västerås, Sweden
- Primary focus is Pingdom product: back end, infrastructure, and front end

This is Pingdom

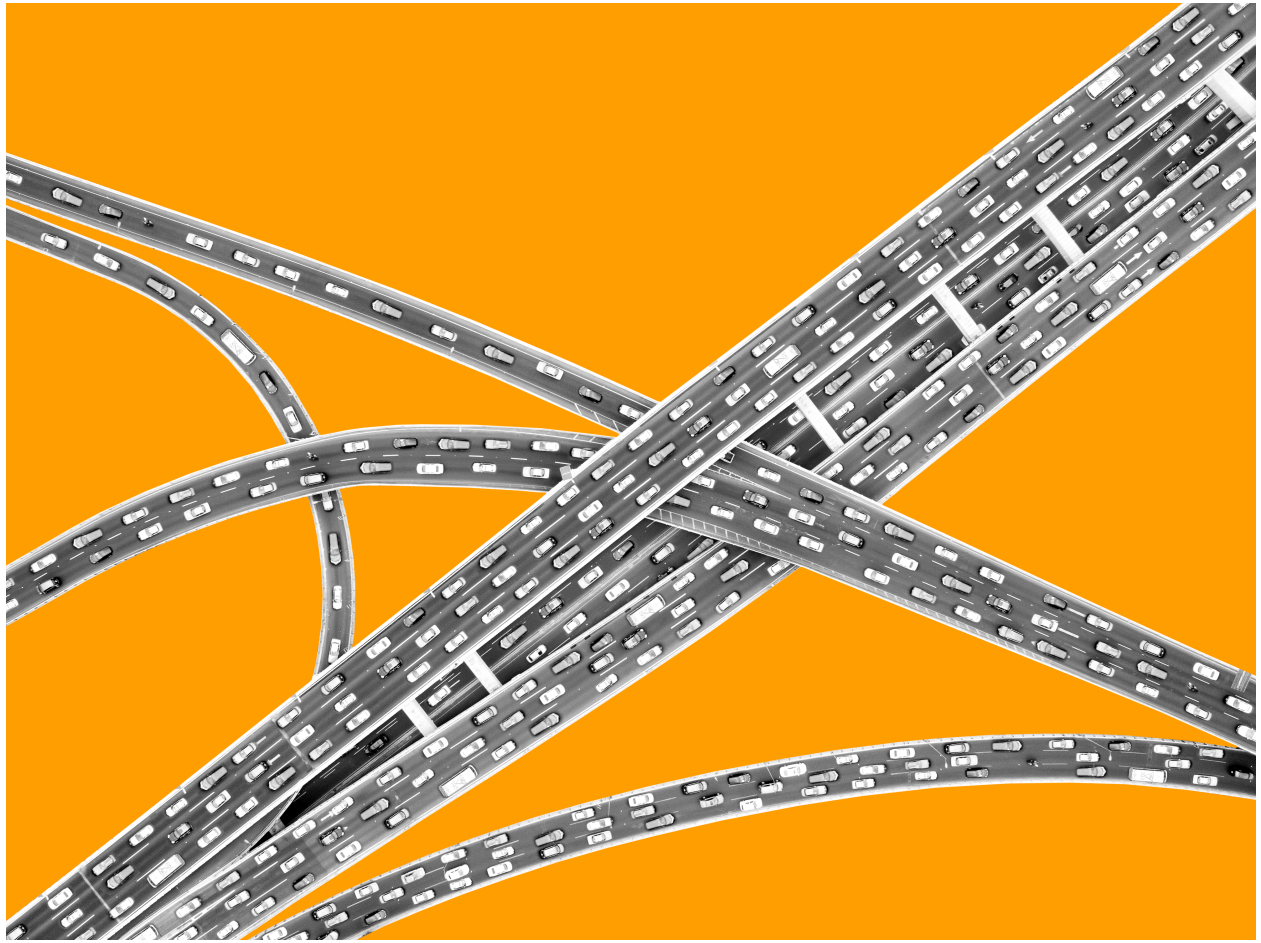


Infrastructure requirements

- Different services have different hardware requirements (instance type and count)
- Multiple database clusters with different hardware requirements
- Multiple event processing system (streaming data, message queues)
- Different services are deployed in different networks with different firewall rules
- Each service has its own set of permissions to access other services (databases, message queues, etc.)
- APIs located behind load balancers
- All logs are forwarded to a specific log management system
- Monitoring and alerting

Automate or fail

- Manual management is bound to fail: human factor
  - Misconfiguration
  - Doing tasks in the wrong order, etc.
  - Non-existing or out-of-date documentation
- Lots of repetitive work
- Risk of snowflake servers/configuration drift
- Risk of misconfigured or missing monitoring/alerting



Enter automation

solarwinds

- Configuration management tools (Chef®, Puppet®, SaltStack®, etc.)
  - Usually used for server provisioning (installing patches, libraries, daemons, etc.)
  - Generation of configuration files
- Orchestration/provisioning tools (Terraform®, CloudFormation)
  - Usually used for hardware provisioning (instance type, number of instances, etc.)
  - Keeps a stored state of the system
  - Allows for trivial referencing across systems (e.g., map a database cluster to a set of application nodes)

Declarative or imperative

- Declarative
  - Describes the desired state
  - Immutable infrastructure
- Imperative
  - Describes how to achieve the desired state
  - Can lead to config drift

solarwinds

Example (made up)

```
- launch_instances:

    count: 10

    image: ami-xyz

    instance_type: hw.type.x
```

solarwinds

Example (made up)

```
resource "hw_instance" "example" {

 count = 10

 image = "ami-xyz"

 instance_type = "hw.type.x"

}
```

Declarative - continued

solarwinds

Example (Terraform)

```
resource "aws_security_group" "collector" {
  name   = "${var.pingdom_environment}-a-collector"
  vpc_id = "${var.vpc}"
  ingress {
    from_port       = ${from_port}
    to_port         = ${to_port}
    protocol        = "TCP"
    security_groups = ["${aws_security_group.collector_elb.id}"]
  }
}
module "ecs_collector" {
  source              = "../ecs"
  pingdom_environment = "${var.pingdom_environment}"
  load_balancers      = ["${aws_elb.collector_elb.name}"]
  num_instances       = "${var.collector_instance_count}"
  instance_type       = "${var.collector_instance_type}"
  security_groups     = ["${aws_security_group.collector.id}"]
  subnets             = ["${var.public_in_subnets}"]
}
```

Infrastructure as code

- Easy (fast) to reproduce a system
- Version controlled infrastructure
- Safe rollbacks
- Easy to collaborate
  - Perform changes and create a PR
  - Share modules between teams
- Infrastructure can be unit tested
- Easy to manage multiple environments
- Safe updates to infrastructure – dry run

Monitoring and alerting

- Monitoring agents configured along with the infrastructure
- Alerting for both infrastructure and products
  - Different alert policies
  - Different alert targets

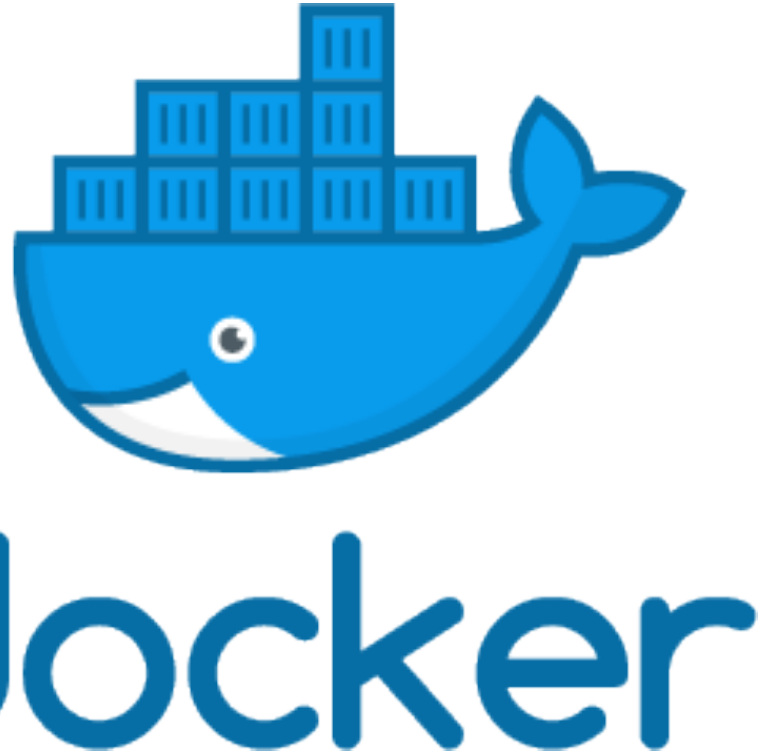
What works for us

- SRE team owns the infrastructure
- Developers collaborate by creating PR when needed. SRE reviews
- Terraform for static resources
  - Reusable modules - one per service
  - Per environment overrides (e.g., hardware size and count)
- SaltStack for services (e.g., databases)
- Homogenous cluster for product deployments
  - Capture dependencies in containers

How we manage products

solarwinds

- Clear separation of modules and products
- Module is a self contained container
  - Containers are stateless
  - Configuration is added via environment variables
  - Scale up by deploying more container instances
- Each module has its own development life cycle
  - Peer review
  - Integration tests



Products

solarwinds

- A product is mainly metadata
  - What container to use?
  - Version of container
  - Configuration for each container
  - Credentials not part of configuration (added later)
- Products are version controlled
  - Peer review
  - Rollbacks, etc.
- Each product has its own development lifecycle

Deployment

solarwinds

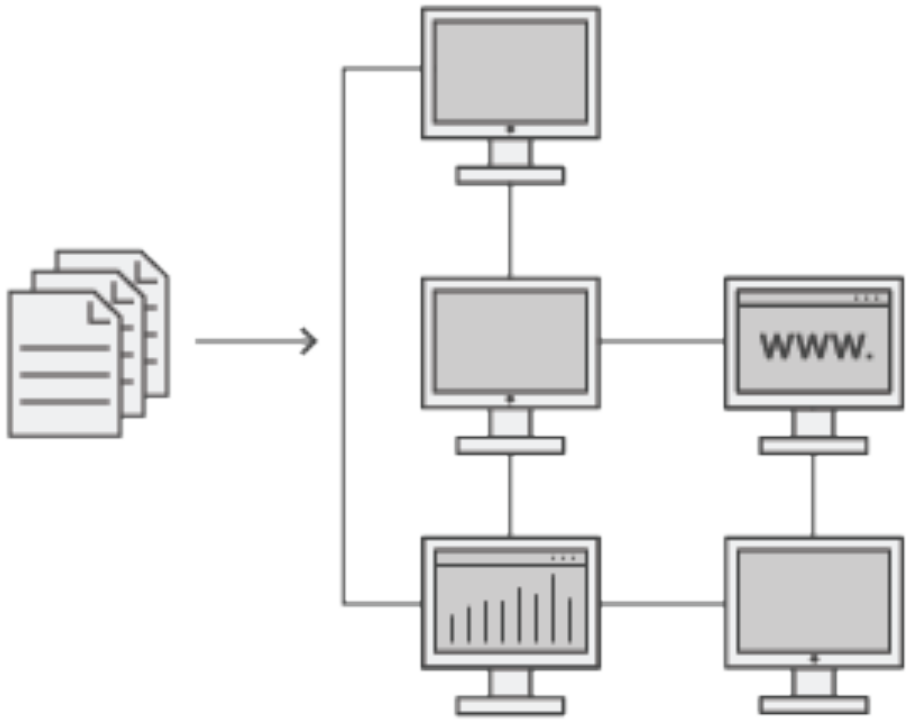
- Deployment is a single function with two arguments
  - Environment (production, system-test, etc.)
  - Product (including version)
- Deployment automatically performs the following tasks:
  - Choose hardware cluster to deploy to
  - Create a deployment artifact with necessarily credentials
  - Push artifact to chosen cluster
  - Spin up the product

Monitoring

- Per-product monitoring may need to be added by developers
  - Metrics
  - Use a log management system than can create alerts from logs
- APM (AppOptics™) can auto generate valuable metrics
- External monitoring to keep track of customers experience (Pingdom®)

Pros

- Infrastructure as application code
  - Agile
  - Iterative
  - Continuously improved infrastructure
- Dare to experiment—learn from failures
  - Rollback is easy, remember?



Caveats

- Naming is important, namespace your components
- Stricter requirements for product development (such as 12-Factor App)
- Don’t reinvent the wheel—use a mature CI & CD system
- Start small, think big

That’s it!

solarwinds

Thank you!