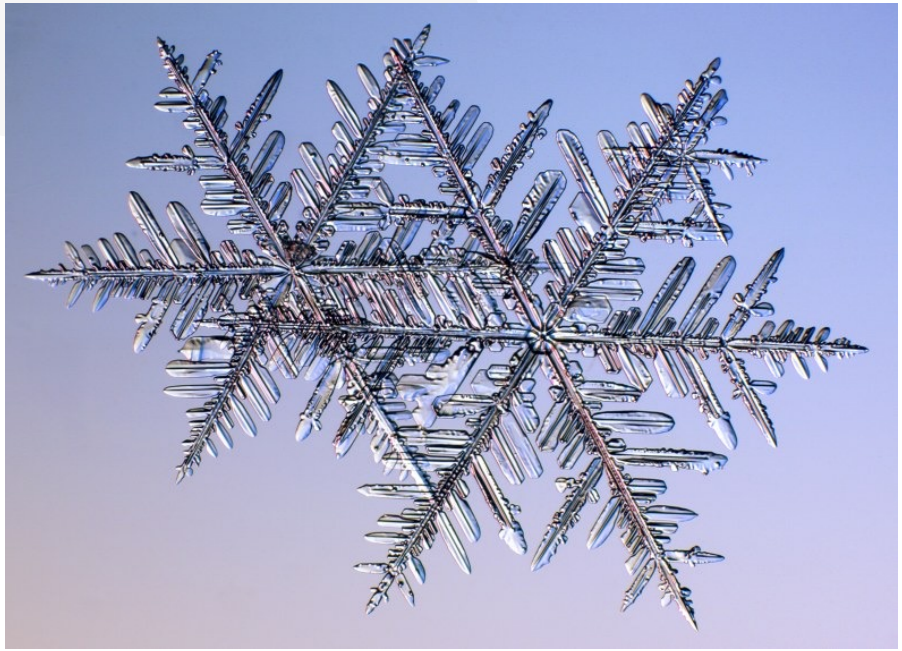
OPITZ CONSULTING

INTRODUCTION IAC

ANTI-PATTERN: SNOWFLAKE SERVERS



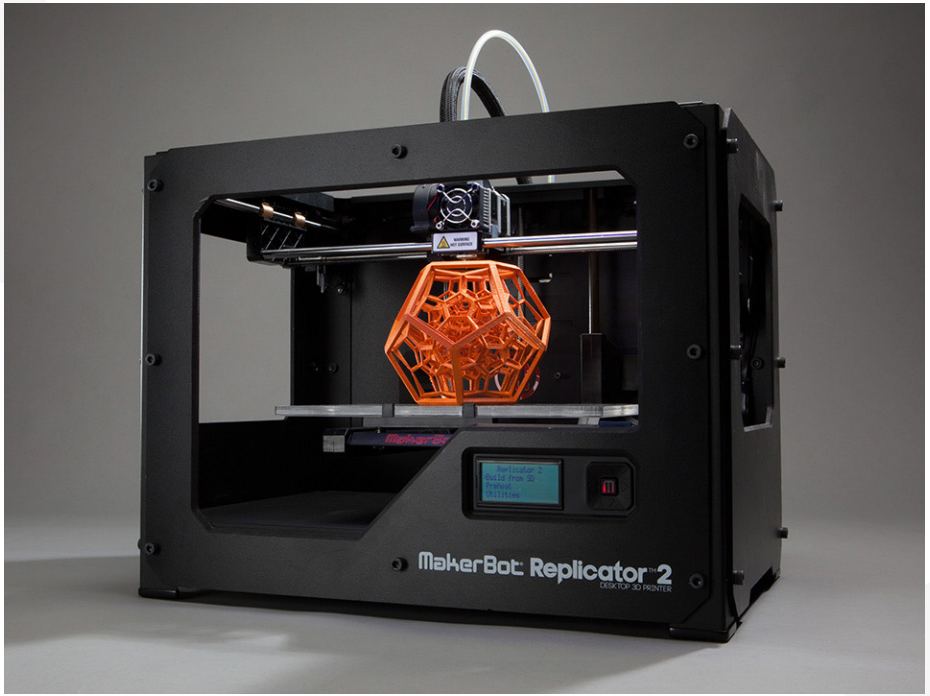
- Every server looks different
  - Different configuration
  - Different software versions
  - Different hardware infrastructure
- Consequence
  - High maintenance effort
  - Requires expert knowledge
  - Error hardly reproducible

PARADIGM SHIFT: MANUAL → AUTOMATED

- Names like kitty.acme.com
- Unique, every configuration is slightly different
  - Line endings, comments, etc.
- Repaired and analyzed in the event of a fault

- Names like vm0042.acme.com
- Configuration is the same on all servers
  - Templating, etc.
- Demolition and new construction in the event of a fault

INFRASTRUCTURE AUTOMATION HELPS



Blueprints for server instances

WELL-KNOWN AUTOMATION TOOLS

WHY NOT JUST USE A SHELL SCRIPT?

AUTOMATION TOOLS OFFER...

- Infrastructure definition at **a high level of abstraction**
- **Modularization concept**
- **Reproducibility** on any number of servers
- A fully **automated process**
- **Idempotence** (maintenance of a target state)
- **Infrastructure as Code**: Versionable and reusable

WHAT IS IAC?

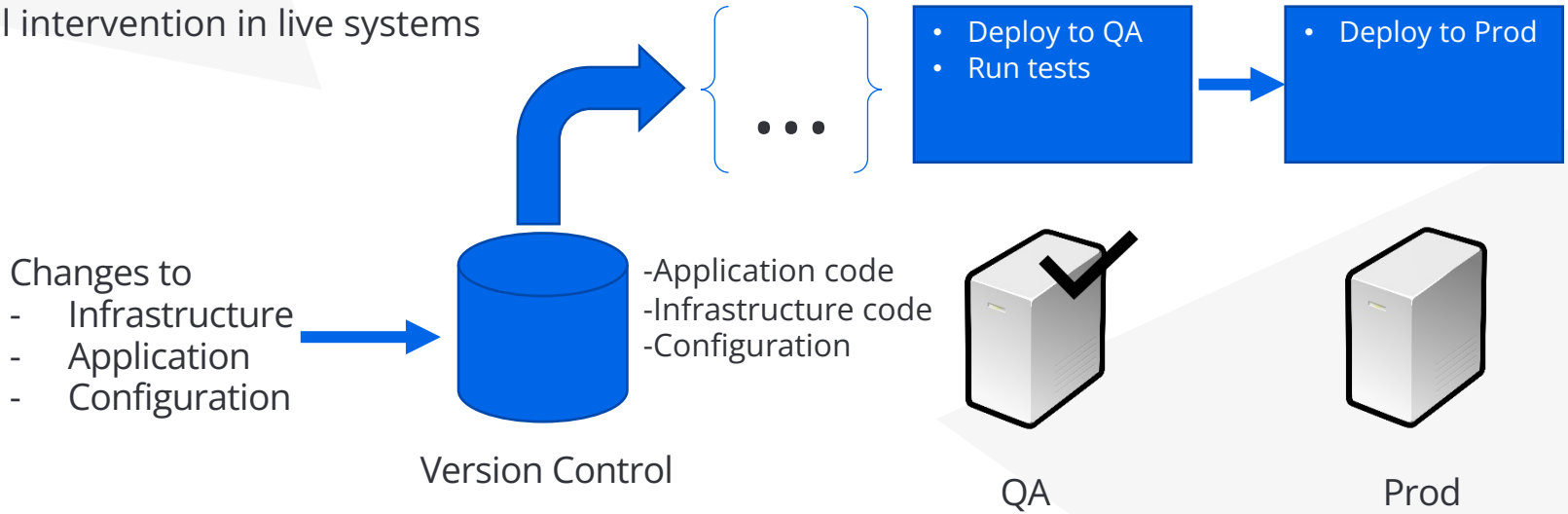
Example: Ansible

```
---
- name: Install httpd
  yum:
    name: httpd
    state: present
- name: Start and enable Service
  service:
    name: httpd
    enabled: yes
    state: started
- name: Setup httpd.conf
  template:
    src: httpd.conf.j2
    dest: /etc/httpd/httpd.conf
```

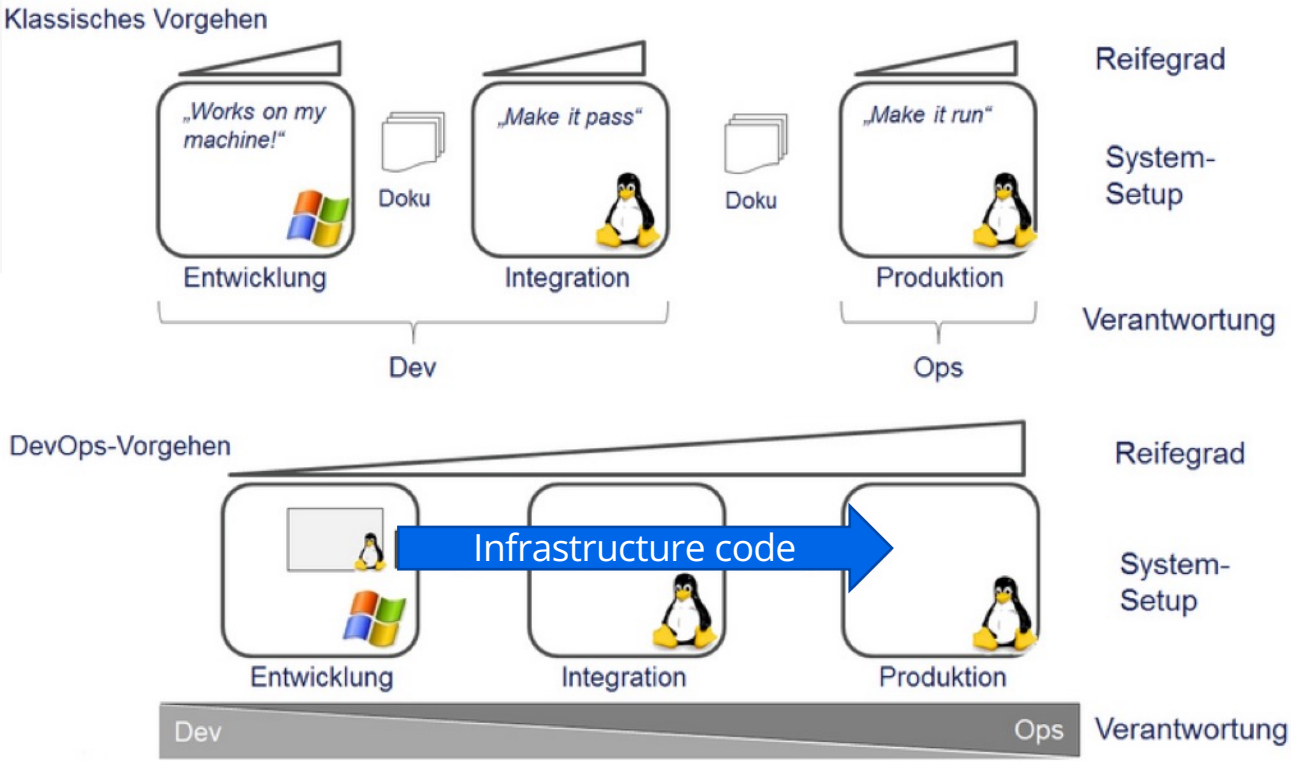
- Executable through automation tool
- Describes a target state
  - Automation tool knows how to achieve this
- Documentation
- Versionable (GIT, etc.)
- Reproducible at any time

QUALITY ASSURANCE THROUGH CONTINUOUS DELIVERY PRACTICES

- Every infrastructure change is triggered by a code commit to a source repository
- Delivery pipeline (e.g. GitHub, GitLab, ...) listens for changes and triggers rollouts
- Verify infrastructure code rollout in test environment first (own infrastructure code!)
- No manual intervention in live systems



CLASSIC APPROACH VS. DEVOPS APPROACH



MOST IMPORTANT BUSINESS DRIVERS

- Transparency and knowledge distribution
  - All system definitions in one place
  - System definition is uniformly structured
  - Modularization lowers the learning curve
- Maintainability
  - Reduced vertical integration due to tool components provided
  - Changes are versioned, traceable and revisable

MOST IMPORTANT BUSINESS DRIVERS

- Reproducibility
  - Infrastructure code ensures identical environment structure
    - Test, Integration, Prod
- Time-to-market
  - System setup "at the touch of a button", as soon as automated
  - Solution components can be reused

SIMPLY(?) START

- Infrastructure automation can / should be started from operation!
  - Acceptance must be present in the company! → Leads to trust
- First step:
  - Map existing infrastructure retrospectively using infrastructure code
  - Only map new systems with IaC
- Added value
  - A quick start to infrastructure automation
  - System setup is documented
  - Entry hurdle for new colleagues is lowered