# How good is your DevOps?

Having “good DevOps” isn’t about having any one tool or practice in place, it takes many elements and people working together.

I’ve developed this scorecard over 7 years with input from customers, employers, and peers; including presenting and discussing it at industry meetups and DevOpsDays conferences. Use it to assess your DevOps and uncover opportunities to fine-tune.

This list is intended as an end-goal for any organization, however most will never achieve all of these due to particular staff, funding, or security constraints or considerations. With the combined effort of your engineering teams, and the support of management and the wider organization, you should be able to achieve most of the following and enjoy the many benefits of sound DevOps.

Have your score calculated automatically - take the quiz on Typeform:

# The DevOps Scorecard

(*1 point for less-important things, and typically up to 5 points for important items, with a rare 10*)

## Foundational

Our DevOps practice is good because:

- ☐ All environments are built for zero-downtime (5 points)
- ☐ All infrastructure is contractual, repeatable and reliable (eg: Infrastructure as Code (IaC) tools like Terraform / CloudFormation) (4 points)
- ☐ No Infrastructure or SaaS Services is/are set up manually (2 points)
- ☐ We use the best tool for the job for each individual concern, rather than trying to use one tool for every job (2 points)
- ☐ No (or minimal) tools have overlapping responsibilities (3 points)
- ☐ Every aspect of an environment has resiliency, redundancy, health checks, monitoring, alerting, auto-healing capabilities and backups (5 points)
- ☐ All applications are (as) self-contained as possible without being dependent on other applications to minimize complexity and potential downtime (3 points)
    - ☐ Where dependencies occur, especially external ones (eg: SaaS cloud services), service can tolerate limited or no availability of such services as much as possible. (3 points)
    - ☐ Circuit breakers or feature flags are in place to detect and act/react accordingly if certain features or dependent services are down (2 points)
    - ☐ Users are notified of such failures of systems or services gracefully (2 points)
- ☐ Secrets are ***never*** in any codebases or in your code history (10 points)
- ☐ Secrets & private data are never stored in logs (5 points)
    - ☐ The possible presence of secrets in logs is regularly audited (2 points)
- ☐ Secrets are managed securely and centrally (Eg: HashiCorp Vault / AWS KMS) (5 points)

## Onboarding

New employee spin-up time is minimal because:

- ☐ Codebases all have coding standards (3 points)
    - ☐ Codebases have automatic forced lint checks to enforce coding standards (3 points)
- ☐ Codebases all have good documentation both in a README and inline code (3 points)
- ☐ Getting access to various systems, codebases, portals, etc to do their job is not a complex task. (3 points)
    - ☐ Ideally all delegated and granted access through a centralized authentication and access system (eg: OAuth, LDAP) (2 points)
- ☐ Our organization has up-to-date diagramming / flowcharting / Entity Relationship Diagrams / wireframes for both higher-level infrastructure/architecture and per-service (10 points)

- [ ] The CI/CD files that deploy services are easy to understand and are accessible and iterable for all engineers (5 points)

## Services and environments

New service creation is simple due to:

- Defined company-wide standards and documentation for:
  - [ ] Code (language-specific) (3 points)
  - [ ] Automation (language-specific) (3 points)
  - [ ] Service deployment (3 points)
- [ ] Minimal effort required to create and deploy new services (3 points)
  - [ ] Allowing and encouraging developers to submit changes to infrastructure (via IaC) (1 point)

Services are built for reliability, maintainability and scalability by ensuring that:

- [ ] All software is built for high-availability (5 points)
- [ ] Requests which need processing time are delegated to a background worker and/or task engine technology (3 points)
- [ ] Code is maintainable (see “Code” below) (3 points)
- [ ] Code is modular (see “Code” below) (3 points)
- [ ] Testing is built into the codebase which is used automatically by CI, and is required to run before merges are accepted (3 points)
  - [ ] Testing is easy for developers to run locally, often with Makefiles or Docker. It is also documented to increase accessibility to new engineers (3 points)
- [ ] Service deployments are made automatically from CI with definition files stored in the codebase (5 points)
- [ ] Deployments are configurable per service, environment, deployment, developer (3 points)

All environments/services are built securely by:

- [ ] Either the infrastructure setup itself, proxies/gateways or services themselves force all traffic to use HTTPS / SSL, with no exception (10 points)
- [ ] All endpoints and all nodes are behind firewalls, only the minimal endpoints are revealed to the public (eg: Security Groups, Network ACLs, Network Policies, Firewalls, etc) (5 points)
- [ ] Service and servers are set up and designed with least-privilege principles (2 points)
- [ ] Roles assigned to services are all least-privilege (eg: IAM / Service Accounts) (5 points)
- [ ] Access to dependant services is created with least-privilege (eg: database access, not using a superuser for the database) (2 points)
- [ ] Using cloud provider(s) best-practice technologies/methodologies/services as much as possible, minimizing tech debt and responsibilities (5 points)
- [ ] All actions, namely super-user ones, leave an audit trail (eg: Services / Users) (3 points)
  - [ ] These audit trails are not editable by anyone or anything (2 points)
- [ ] Environments and services are architected based on gathered use-cases/requirements and team skill-sets, not a line drawn in the sand or an abstract/hard requirement (1 point)

- [ ] Environments are as simple as they can be, using the least amount of technologies possible while still accomplishing the goal (KISS) (2 points)

## Code

- Code is maintainable:
  - [ ] Uses language formatting/styling best-practices and standards (per-language) (2 points)
  - [ ] Has documentation on how to quickly get familiar with the codebase and get a development environment working (2 points)
  - [ ] Company emphasis on knowledge sharing / team pairing / etc (3 points)
  - [ ] Has automated testing, the more the merrier (unit, end to end, acceptance, load, etc) (3 points)
  - [ ] Uses automatic Continuous Integration (CI) techniques/technologies to automatically verify code is within standards and passes all tests before allowing merges and/or deploys (5 points)
  - [ ] Uses automatic Continuous Deployment (CD) techniques to automatically deploy code, enabling automated (or manual) quality or integration tests afterwards (5 points)
  - [ ] All code is kept DRY ("Do not Repeat Yourself"). Library code is in a shared codebase / module if needed for re-use across multiple projects. (5 points)
  - [ ] KISS. Keep It Simple, Stupid. Everything in your infrastructure and development is designed to be accessible, is documented, is kept as simple as it can be but fulfilling the requirements it needs to (5 points)
- Code is modular:
  - [ ] Pieces of the codebase are engineered with modularity in mind (5 points)
  - [ ] Libraries should be able to be re-used in other projects as a simple drop-in (3 points)

## Documentation

Documentation & diagramming are available and maintained for:

- [ ] Infrastructure (5 points)
- [ ] Codebases (3 points)
- [ ] Services and service (inter)dependencies (eg: flowcharts) (3 points)
- [ ] Deployment and rollback workflow (5 points)
- [ ] Edge cases (5 points)
- [ ] Disaster Recovery (10 points)

## Deployments

Good DevOps practices from above ensures that:

- [ ] Infrastructure deployments and modifications are fully automated/scripted

- [ ] Engineers do not use command line to apply changes to any environment (optional, highly opinionated and customer / regulatory dependant) (1 point)
- [ ] Code deployments are automated (10 points)
    - [ ] Optionally with a manual circuit breaker for live deploys to wait for manual intervention/approval (1 point)
- [ ] Codebases are split into separate repositories per service, microservice, purpose (5 points)

- [ ] All code is in source code management (preferably git) (10 points)
    - [ ] Using industry-standard branching/merging and releases, ideally company-wide (5 points)
        - See: https://semver.org
        - See: https://datasift.github.io/gitflow/IntroducingGitFlow.html
    - [ ] Senior engineer(s) or equivalent reviews code (5 points)
    - [ ] Maintainer / owner / peer(s) approve before merge (after above review) (5 points)
    - [ ] Squashing code from Merge Requests (optional) (1 point)
    - [ ] Remove branches after merge (2 points)
- [ ] Service deployments use zero-downtime mechanisms (such as Blue/Green or Canary) (5 points)
- [ ] Deployments are painless, single-click or fully automated (5 points)
- [ ] Rollbacks are painless, single-click or fully automated (3 points)
    - [ ] Rollback process is well-documented (1 point)
    - [ ] Test/validate rollbacks before going live with any new service (2 points)
- [ ] Software release process focuses on small deployments (eg: not sprint deploys) (3 points)

## Testing

Good DevOps practice has:

- [ ] Robust tests for any/all aspects of code or services (IaC / backend / API / frontend) (3 points)
- [ ] Automated unit / acceptance testing (2 points)
- [ ] Automated end to end / integration testing (3 points)
- [ ] Automated performance (load/soak) testing, possibly on every major release candidate (1 point)

## Monitoring, logging and debugging

Good DevOps practice has:

- [ ] Multi-faceted monitoring solution in place, monitoring typically USE/RED metrics and/or traditional server metrics where relevant (5 points)
- [ ] Custom application monitoring & metrics in place to detect sub-service failures / outages (2 points)
- [ ] All services logging to a centralized logging platform, with read-only access given to engineers necessary to perform debugging/maintenance/etc (5 points)

- [ ] A standardized log format (eg: JSON) used for parsability in a centralized logging platform (2 points)
- [ ] Access for all developers to all systems even on production to monitor, maintain and debug (3 points)
    - [ ] Restrict role/group based access to administration and monitoring portals to only specific components/areas where needed and using least-privilege (3 points)
- [ ] No TRACE/DEBUG logging on Production environments (3 points)

## Score

There's a total possible score of 305 points for all items. It's a good idea to note the date and your score then re-do the checklist at a later date to see the impact of your optimizations.