50 AI Prompts to Automate Everything You Hate as a DevOps Engineer

DevOps automation with AI

Zudonu Osomudeya • 20 min read • 2025-08-11

s://medium.com/@osomudeyazudonu/50-ai-prompts-to-automate-everything-you-hate-as-a-devops-engineer-672e71a99d

NOT A MEDIUM MEMBER? READ FULL STORY HERE

The Day I Counted My Tools (And Questioned My Life Choices)

Last month, I did something that probably qualifies as masochistic: I started listing every tool in a typical modern DevOps stack. Not just the big ones, Kubernetes, Jenkins, Terraform, but *everything*. The monitoring tools, the logging aggregators, the secret managers, the CI/CD platforms, the container registries, the service meshes, the... you get the idea.

I stopped counting after reaching **dozens and dozens** of tools.

I realized I needed a spreadsheet just to track my spreadsheets. We have tools to monitor our monitoring tools. We use Helm/Kustomize to deploy operators that manage custom resources that configure other tools that... honestly, the complexity becomes fractal at some point.

And then it hit me like a poorly configured load balancer: I was spending more time learning tools than actually solving problems.

Don’t get me wrong, I love technology. I love the elegance of a well-orchestrated deployment, the satisfaction of a perfectly tuned autoscaler, the zen-like peace of green monitoring dashboards. But somewhere along the way, we’ve built ourselves into a corner of complexity that would make a Rube Goldberg machine look simple.

The worst part? Most of my day wasn’t spent on the interesting challenges like the architecture decisions, the performance optimizations, the creative problem-solving that drew me to this field. Instead, I was writing the same bash scripts over and over, copy-pasting YAML configurations, explaining the same concepts in documentation that nobody would read, and debugging issues that felt like deja vu.

Then, on a particularly frustrating afternoon wrestling with a Terraform configuration that seemed determined to spite me, I had a thought: “What if I just... asked for help?”

Not from a colleague (they were all busy fighting their own infrastructure battles), but from the AI chatbot I’d been ignoring in my browser tab.

“Write me a Terraform module for a secure S3 bucket with versioning and encryption,” I typed, feeling slightly ridiculous.

Thirty seconds later, I had clean, documented, production-ready code.

I stared at my screen, processing what had just happened. In half a minute, AI had done what would have taken me at least 20 minutes of documentation-diving, stack-overflow-searching, and configuration-tweaking.

That was the moment I realized I’d been doing DevOps wrong.

The Irony

Here’s the irony that keeps me up at night: We’re the automation people. We write scripts to automate deployments. We create pipelines to automate testing. We build infrastructure as code to automate provisioning. We automate *everything*... except our own repetitive daily tasks.

We’ll spend three days automating a deployment that runs once a week, but we’ll manually write the same kubectl commands every single day. We’ll architect elaborate CI/CD pipelines, but we’ll copy-paste bash scripts like it’s 1999. We’ll create sophisticated monitoring dashboards, but we’ll spend hours manually parsing logs when something goes wrong.

The reason is simple: automating infrastructure is satisfying. It’s creative. It’s the kind of work that gets you promoted.

Automating the mundane stuff, the documentation, the boilerplate code, the repetitive troubleshooting, feels like admitting defeat. Like we should be above needing help with “simple” tasks.

But here’s the thing: AI doesn’t judge you for asking how to write a regex that actually works. It doesn’t roll its digital eyes when you need help with YAML indentation for the thousandth time. It doesn’t make you feel stupid for forgetting the exact syntax of a jq command.

AI is like having a senior engineer who never gets tired, never gets frustrated, and never makes you feel dumb for asking questions. And unlike that one colleague who sighs every time you ask for help, AI actually *enjoys* solving your problems.

The 50 Prompts That Will Change Your DevOps Life

What follows are 50 carefully crafted prompts that will automate the soul-crushing parts of your job. I've organized them into the categories that cause the most pain, tested them in real scenarios, and included examples of when to use each one.

Copy them. Modify them. Make them your own. And watch as the boring parts of DevOps become the automated parts of DevOps.

1. Generate GitHub Actions Workflow

When to use: Starting a new project or migrating from another CI platform.

```
Create a GitHub Actions workflow for a Node.js application that:
- Runs on push to main and pull requests
- Tests on Node versions 16, 18, and 20
- Runs lint, test, and build steps
- Only deploys to production on main branch merges
- Includes proper caching for node_modules
- Sets up environment-specific secrets
```

2. Debug Pipeline Failures

When to use: When your pipeline fails and the logs look like hieroglyphics.

```
I have a CI/CD pipeline failure. Here's the error log:
[paste your error log]

Help me:
1. Identify the root cause
2. Provide specific steps to fix it
3. Suggest ways to prevent this error in the future
4. Add appropriate error handling
```

3. Optimize Pipeline Performance

When to use: When your 5-minute build somehow became a 30-minute ordeal.

```
Analyze this pipeline configuration and suggest optimizations:
[paste your pipeline config]

Focus on:
- Parallel job execution
- Caching strategies
- Unnecessary steps
- Build time reduction
- Resource efficiency
```

4. Create Multi-Environment Pipeline

When to use: Setting up proper dev/staging/prod workflows.

```
Design a CI/CD pipeline strategy for deploying to:
- Development (auto-deploy on feature branch push)
- Staging (auto-deploy on main branch merge)
- Production (manual approval required)

Include:
- Environment-specific configurations
- Rollback strategies
- Security considerations
- Approval workflows
```

5. Generate Pipeline Documentation

When to use: When your team keeps asking “how does this pipeline work?”

```
Create comprehensive documentation for this CI/CD pipeline:
[paste your pipeline configuration]

Include:
- Overview and purpose
- Trigger conditions
- Step-by-step breakdown
- Environment variables needed
- Troubleshooting guide
- How to modify for new requirements
```

6. Parse Complex Log Files

When to use: When grep isn’t cutting it and you need to extract meaningful data.

```
Help me parse this log file to extract:
[specify what you need: error patterns, performance metrics, user actions, etc.]

Log sample:
[paste a representative sample of your logs]

Provide:
- Command-line tools (awk, sed, jq) for extraction
- Regex patterns for matching
- One-liner commands for common queries
- Script to automate the parsing
```

7. Create Log Analysis Scripts

When to use: For recurring log analysis tasks that eat up your time.

```
Create a script that analyzes [application/service] logs to:
- Count error types and frequency
- Identify performance bottlenecks
- Extract timing information
- Generate summary reports
- Alert on specific patterns

Input format: [describe your log format]
Output: Human-readable summary + CSV for further analysis
```

8. Build Monitoring Queries

When to use: Writing Prometheus, Grafana, or CloudWatch queries from scratch.

```
Create monitoring queries for [Prometheus/CloudWatch/Grafana] to track:
- [specific metric you want to monitor]
- Alert when [condition]
- Dashboard visualization showing [what you want to see]

Context: [describe your application/infrastructure]
Current metrics available: [list existing metrics if known]
```

9. Design Alert Strategies

When to use: When you're drowning in false positives or missing real issues.

```
Design an alerting strategy for [describe your system] that:
- Minimizes false positives
- Catches real issues before users notice
- Provides actionable information
- Escalates appropriately
- Includes runbook links

Current pain points: [describe what's not working]
Team size: [number] people
On-call rotation: [describe your setup]
```

10. Troubleshoot Monitoring Issues

When to use: When your monitoring tools are lying to you.

```
My monitoring is showing [describe the issue], but [describe what's actually
happening].

Help me:
1. Debug why the metrics don't match reality
```

```
2. Fix the monitoring configuration
3. Identify blind spots in current monitoring
4. Add missing metrics/alerts
5. Validate the monitoring is working correctly
Monitoring stack: [list your tools]
```

11. Generate API Documentation

When to use: When you need to document an API and make it actually useful.

```
Create comprehensive API documentation for:
[paste your API specification/swagger/code]

Include:
- Clear endpoint descriptions
- Request/response examples
- Authentication requirements
- Error codes and handling
- Rate limiting information
- SDK examples in [preferred languages]
- Common use cases and workflows
```

12. Write Runbook Documentation

When to use: For those 2 AM incidents when you can’t remember how to fix things.

```
Create a troubleshooting runbook for [service/application] covering:
- Common failure scenarios: [list them]
- Step-by-step recovery procedures
- Required access/permissions
- Escalation contacts
- How to communicate with stakeholders
- Post-incident cleanup steps

Make it usable by junior engineers during high-stress situations.
```

13. Create Architecture Documentation

When to use: When explaining your system architecture to new team members.

```
Document this system architecture:
[describe your system, paste diagrams, or provide code]

Include:
- High-level overview
- Component responsibilities
- Data flow diagrams
- Integration points
- Scalability considerations
- Security boundaries
- Deployment topology
- Decision rationale for key choices
```

14. Write Installation Guides

When to use: When setting up your application shouldn't require a computer science degree.

```
Create step-by-step installation documentation for [your application/service]
that includes:
- Prerequisites and system requirements
- Installation methods (Docker, package manager, source)
- Configuration options
- Verification steps
- Common troubleshooting
- Upgrade procedures
- Uninstall instructions

Target audience: [describe skill level]
```

15. Generate README Files

When to use: For every repository that deserves better than "TODO: Add documentation."

```
Create a comprehensive README for this project:
[paste repository information, code samples, or describe the project]

Include:
- Clear project description and purpose
- Quick start guide
- Installation instructions
- Usage examples
- Configuration options
- Contributing guidelines
- License information
- Badges for build status, coverage, etc.
```

16. Debug Application Errors

When to use: When error messages are about as helpful as a chocolate teapot.

```
I'm getting this error in [application/service]:
[paste full error message and stack trace]

Context:
```

```
- What I was trying to do: [describe the action]
- Environment: [production/staging/local]
- Recent changes: [any deployments, config changes, etc.]
- Additional symptoms: [other weird behavior]

Help me:
- Understand what this error means
- Find the root cause
- Fix it step by step
- Prevent it from happening again
```

17. Performance Troubleshooting

When to use: When everything was fast yesterday but slow today.

```
My [application/database/service] performance has degraded. Help me investigate:

Symptoms:
- [describe what's slow]
- Started around: [timeframe]
- Affects: [users/operations/etc.]

Current metrics:
- [paste relevant performance data]

System info:
- [infrastructure details]
- [recent changes]

Provide a systematic troubleshooting approach with specific commands to run.
```

18. Database Query Optimization

When to use: When your database queries are slower than a government website.

```
Optimize this database query:
[paste your SQL query]

Current performance:
- Execution time: [time]
- Rows examined: [number]
- Database: [MySQL/PostgreSQL/etc.]

Table schemas:
[paste relevant table structures]

Provide:
- Optimized query
- Index recommendations
- Explanation of improvements
- Alternative approaches
```

19. Container Troubleshooting

When to use: When Docker containers are misbehaving in creative ways.

```
My container is [crashing/not starting/behaving weird]:

Dockerfile:
[paste Dockerfile]
Error logs:
[paste container logs]
Environment:
- Docker version: [version]
- Host OS: [OS]
- Kubernetes version: [if applicable]
Help me:
1. Identify the issue
2. Fix the container configuration
3. Add proper health checks
4. Optimize the image
```

20. Network Connectivity Issues

When to use: When services can't talk to each other and you're playing network detective.

```
I have a network connectivity issue:
- Service A ([details]) can't reach Service B ([details])
- Error: [paste error message]
- Network setup: [describe topology]
- Security groups/firewalls: [describe rules]

Help me:
1. Debug the connection step by step
2. Identify where the traffic is blocked
3. Provide specific commands to test connectivity
4. Fix the networking configuration
```

21. Generate Deployment Scripts

When to use: When you need bulletproof deployment automation.

```
Create a bash deployment script for [describe your application] that:
- Pulls latest code from git
- Builds the application
- Runs tests before deploying
- Creates backups before deployment
- Deploys with zero downtime
- Verifies deployment success
- Rolls back on failure
- Sends notifications

Include proper error handling and logging.
```

22. System Maintenance Scripts

When to use: For those routine tasks that you always forget to do.

```
Create a system maintenance script that:
- Cleans up old log files (keep last 30 days)
- Removes unused Docker images and containers
- Updates packages securely
- Checks disk space and alerts if >80% full
- Backs up critical configurations
- Generates maintenance report

Make it safe to run automatically via cron.
```

23. Log Rotation and Cleanup

When to use: When your disk space disappears faster than your weekend.

```
Create a log management script that:
- Rotates logs for [specify your applications]
- Compresses old logs
- Deletes logs older than [timeframe]
- Handles multiple log formats
- Preserves logs during active debugging
- Sends alerts when cleanup fails

Applications: [list your apps and their log locations]
```

24. Backup and Restore Scripts

When to use: Because backups are like insurance, boring until you need them.

```
Create backup and restore scripts for:
- Database: [type and connection details]
- Application files: [locations]
- Configuration files: [locations]
- User data: [locations]
```

```
Requirements:
- Encrypted backups
- Remote storage (S3/GCS/etc.)
- Retention policy: [specify]
- Verification that backups work
- Easy restore process
- Progress monitoring
```

25. Health Check Scripts

When to use: For proactive monitoring that actually prevents problems.

```
Create a comprehensive health check script for [your infrastructure] that:
- Tests all critical services
- Verifies database connections
- Checks API endpoints
- Monitors resource usage
- Validates SSL certificates
- Ensures backup processes are working
- Generates readable status reports

Exit codes for integration with monitoring systems.
```

26. Generate Basic Deployment Manifests

When to use: When you need K8s manifests but don’t want to memorize the API.

```
Create Kubernetes manifests for [describe your application]:
- Deployment with [number] replicas
- Service (ClusterIP/LoadBalancer/NodePort)
- ConfigMap for configuration
- Secret for sensitive data
- Ingress for external access
- Resource limits: [specify CPU/memory]
- Health checks: [readiness/liveness probes]

Application details:
[provide image, ports, environment variables, etc.]
```

27. Create Advanced Workload Configurations

When to use: For production-ready applications with all the bells and whistles.

```
Create production-ready Kubernetes manifests for [application] including:
- Horizontal Pod Autoscaler
- Pod Disruption Budget
- Network Policies for security
- Init containers for setup
- Persistent volumes for data
- ServiceMonitor for Prometheus
- RBAC configurations
- Multi-environment support (dev/staging/prod)

Requirements: [specify your needs]
```

28. Generate Helm Charts

When to use: When you want reusable, parameterized Kubernetes applications.

```
Create a Helm chart for [your application] with:
- Parameterized values.yaml
- Template for all K8s resources
- Support for multiple environments
- Optional components (monitoring, persistence, etc.)
- Input validation
- Helpful NOTES.txt
- README with usage examples

Application components: [list deployments, services, etc.]
```

29. Debug Kubernetes Issues

When to use: When K8s is giving you the silent treatment.

```
Help me debug this Kubernetes issue:

Problem: [describe what's not working]
Expected behavior: [what should happen]

Current state:
kubectl get pods: [paste output]
kubectl describe pod [pod-name]: [paste output]
kubectl logs [pod-name]: [paste logs]

Provide:
- Root cause analysis
- Specific kubectl commands to investigate further
- Step-by-step fix
- Prevention strategies
```

30. Optimize Resource Usage

When to use: When your K8s cluster costs more than your mortgage.

```
Analyze and optimize these Kubernetes resources:
[paste your manifests or kubectl get output]

Help me:
- Right-size CPU and memory requests/limits
- Optimize node utilization
- Reduce costs while maintaining performance
- Implement proper autoscaling
- Identify unused or oversized resources
Current cluster info: [node types, utilization, etc.]
```

31. Generate Terraform Modules

When to use: When you need reusable infrastructure components.

```
Create a Terraform module for [infrastructure component] with:
- All necessary resources for [specific use case]
- Input variables for customization
- Output values for integration
- Documentation with examples
- Validation rules for inputs
- Tags for resource management

Provider: [AWS/GCP/Azure]
Requirements: [specify your needs]
```

32. Secure Infrastructure Templates

When to use: When security isn't an afterthought but a requirement.

```
Create secure Terraform configuration for [infrastructure] following best
practices:
- Least privilege access
- Encryption at rest and in transit
- Network segmentation
- Audit logging enabled
- Compliance with [specify standards]
- Secret management
- Backup and disaster recovery

Platform: [cloud provider]
Compliance requirements: [list any specific needs]
```

33. Multi-Environment Infrastructure

When to use: When you need consistent infrastructure across environments.

```
Design Terraform structure for multi-environment deployment:
- Environments: dev, staging, production
- Shared resources: [list what should be shared]
- Environment-specific: [list differences]
- State management strategy
- CI/CD integration
- Cost optimization
- Security boundaries

Provide directory structure and example configurations.
```

34. Terraform Troubleshooting

When to use: When Terraform plans look scarier than a horror movie.

```
Help me troubleshoot this Terraform issue:

Error message: [paste error]
Command run: [terraform plan/apply/etc.]
Configuration: [paste relevant .tf files]

Current state:
- What changed recently: [deployments, config changes]
- Expected outcome: [what should happen]

Provide:
- Root cause explanation
- Step-by-step fix
- Commands to safely resolve
- Prevention strategies
```

35. Cost Optimization Analysis

When to use: When your cloud bill makes you question your life choices.

```
Analyze this Terraform configuration for cost optimization:
[paste your .tf files]

Identify:
- Oversized resources
- Unused resources
- More cost-effective alternatives
- Reserved instance opportunities
- Automation for cost management
- Monitoring for cost alerts
Current monthly spend: [if known]
Primary workloads: [describe usage patterns]
```

36. Practice Interview Questions

When to use: When you want to ace that DevOps interview.

```
Generate DevOps interview questions for a [level] engineer role at a [company
size/type]:
- Technical questions about [specific technologies]
- Scenario-based troubleshooting
- System design problems
- Behavioral questions
- Hands-on challenges

Include:
- Sample answers for technical questions
- Approach strategies for scenarios
- Common follow-up questions
- Red flags to avoid
```

37. Resume Optimization

When to use: When your resume needs to actually get past the ATS robots.

```
Optimize this DevOps resume section:
[paste your experience/skills section]

For role: [paste job description]

Improve:
- Keywords for ATS optimization
- Impact-focused bullet points
- Quantified achievements
- Technical skills presentation
- Action verbs and clarity
- Industry-specific terminology
```

38. System Design Preparation

When to use: For those “design a scalable system” interview questions.

```
Help me design [system type] for [scale/requirements] including:
- Architecture overview
- Technology choices and rationale
- Scalability considerations
- Security measures
- Monitoring and observability
- Disaster recovery
- Cost estimation
- Trade-offs and alternatives
```

```
Interview context: [company type, role level]
```

39. Salary Negotiation Preparation

When to use: When you want to get paid what you’re actually worth.

```
Help me prepare for salary negotiation:
- Current role: [your position]
- Experience: [years in DevOps]
- Location: [city/remote]
- Offer received: [if applicable]
- Market research: [what you've found]

Provide:
- Market rate analysis
- Negotiation talking points
- Value proposition statements
- Benefits to negotiate beyond salary
- Scripts for difficult conversations
```

40. Career Growth Planning

When to use: When you want to level up but don’t know how.

```
Create a career development plan for advancing from [current level] to [target
level] in DevOps:

Current skills: [list your strengths]
Target role: [describe desired position]
Timeline: [your timeframe]
Interests: [cloud platforms, automation, etc.]

Include:
- Skills gap analysis
- Learning roadmap
- Certification recommendations
- Project ideas for portfolio
- Networking strategies
- Timeline with milestones
```

41. GitOps Workflow Setup

When to use: When you want deployments as smooth as your Git commits.

```
Design a GitOps workflow for [application/infrastructure] using
[ArgoCD/Flux/Jenkins]:
- Git repository structure
- Deployment automation
- Environment promotion
- Rollback strategies
- Secret management
- Policy enforcement
- Monitoring integration

Requirements:
- Environments: [list them]
- Approval processes: [describe needs]
- Security requirements: [specify]
```

42. Chaos Engineering Scripts

When to use: When you want to break things on purpose (for science).

```
Create chaos engineering experiments for [your system] to test:
- Network failures
- Service dependencies
- Resource exhaustion
- Data corruption scenarios
- Regional outages

Include:
- Gradual escalation approach
- Safety measures and circuit breakers
- Monitoring and alerting
- Rollback procedures
- Experiment documentation
- Success criteria
Current architecture: [describe your system]
```

43. Security Automation

When to use: When security scanning should be automatic, not optional.

```
Create security automation for [your environment] including:
- Vulnerability scanning in CI/CD
- Secret detection in code
- Container image security
- Infrastructure compliance checks
- Security policy enforcement
- Incident response automation
- Compliance reporting

Current stack: [list your technologies]
Compliance requirements: [SOC2/PCI/HIPAA/etc.]
```

44. Cost Management Automation

When to use: When you want to stop financial surprises in your cloud bill.

```
Design automated cost management for [cloud environment]:
- Resource right-sizing recommendations
- Unused resource identification
- Budget alerts and enforcement
- Cost allocation and tagging
- Reserved instance optimization
- Automated cleanup policies
- Cost reporting dashboards

Current spend: [monthly budget]
Primary cost drivers: [compute/storage/network]
Team size: [for cost allocation]
```

45. Disaster Recovery Automation

When to use: When disasters shouldn't require manual heroics.

```
Create disaster recovery automation for [your infrastructure]:
- Automated backup verification
- Cross-region replication
- Failover procedures
- Data integrity checks
- Recovery time optimization
- Testing automation
- Communication workflows

Requirements:
- RTO: [Recovery Time Objective]
- RPO: [Recovery Point Objective]
- Critical systems: [list priorities]
- Budget constraints: [specify limits]
```

46. Database Migration Scripts

When to use: When you need to move data without losing your sanity.

```
Create database migration strategy from [source] to [destination]:
- Schema migration approach
- Data transfer methods
- Downtime minimization
- Rollback procedures
- Data validation
- Performance optimization
- Testing strategy
```

```
Migration details:
- Data size: [approximate volume]
- Downtime tolerance: [maximum acceptable]
- Critical tables: [list most important]
- Current performance: [baseline metrics]
```

47. API Testing Automation

When to use: When manual testing feels like Groundhog Day.

```
Create comprehensive API testing automation for:
[paste API documentation or describe endpoints]

Include:
- Functional tests for all endpoints
- Performance/load testing
- Security testing (authentication, authorization)
- Contract testing
- Error handling validation
- CI/CD integration
- Reporting and alerting

Testing tools preference: [Postman/REST Assured/etc.]
```

48. Environment Provisioning

When to use: When setting up environments shouldn’t take a PhD.

```
Create automated environment provisioning for [your application]:
- Infrastructure setup (compute, networking, storage)
- Application deployment
- Configuration management
- Data seeding/migration
- Service dependencies
- Monitoring setup
- User access management

Requirements:
- Environment types: [dev/staging/prod/etc.]
- Provisioning time: [target duration]
- Cost optimization: [budget constraints]
- Security requirements: [access controls]
```

49. Performance Testing Framework

When to use: When you want to know your system breaks before your users do.

```
Design performance testing strategy for [your application]:
- Load testing scenarios
- Stress testing approach
- Endurance testing plans
- Spike testing procedures
- Bottleneck identification
- Automated test execution
- Result analysis and reporting

Current system:
- Architecture: [describe components]
- Expected load: [users/requests/etc.]
- Performance requirements: [SLAs/targets]
- Critical user journeys: [list key workflows]
```

50. Incident Response Automation

When to use: When 3 AM incidents need to be less soul-crushing.

```
Create incident response automation for [your environment]:
- Automated detection and alerting
- Incident classification and routing
- Initial response procedures
- Communication workflows
- Evidence collection
- Recovery automation
- Post-incident analysis

Current pain points:
- Mean time to detection: [current MTTD]
- Mean time to recovery: [current MTTR]
- Team size: [on-call rotation]
- Critical services: [list priorities]
- Communication channels: [Slack/email/SMS]
```

Building Your AI-Powered DevOps Habit

Now that you have these 50 prompts, the real magic happens when you start using them consistently. Here’s how to make AI a natural part of your daily workflow:

Start Small, Think Big

Don’t try to use all 50 prompts tomorrow. Pick the three that address your biggest pain points right now. Maybe it’s the log parsing that takes you 30 minutes every morning, or the YAML configurations you keep copy-pasting from Stack Overflow.

Use AI for these tasks consistently for a week. Once it becomes muscle memory, add another prompt to your toolkit.

Customize for Your Environment

These prompts are starting points, not gospel. Modify them to match your specific tools, naming conventions, and requirements. The prompt that asks for “Kubernetes manifests” should become “EKS manifests with our company’s resource naming standards and security policies.”

Create Your Own Prompt Library

As you use these prompts, you’ll discover patterns specific to your work. Maybe you always need Terraform modules with specific tags, or your log analysis always looks for particular error patterns. Create your own prompts for these recurring needs.

Store them somewhere accessible, a notes app, a shared wiki, or even a simple text file. The key is making them easy to find when you need them.

Teach Your Team

The best automation is automation that helps everyone. Share these prompts with your team. Create a shared prompt library. Host a lunch-and-learn about AI-assisted DevOps.

When your junior engineers can use AI to generate production-ready Terraform modules instead of struggling with documentation, everyone wins.

Measure the Impact

Track how much time these prompts save you. Not obsessively, but enough to see the difference. That 30-minute log analysis that now takes 5 minutes? That’s 25 minutes you can spend on interesting problems instead of grep commands.

Over a month, those minutes add up to hours. Over a year, those hours add up to days or weeks of your life back.

The Future of DevOps is Already Here

We’re living through a fundamental shift in how technical work gets done. The boring, repetitive tasks that used to define much of our day are becoming automated. The tools are getting smarter, the prompts are getting better, and the barriers to automation are getting lower.

This isn’t about AI replacing DevOps engineers, it’s about AI making DevOps engineers more human. When you’re not spending half your day writing bash scripts and YAML files, you can spend that time on the creative, strategic work that actually matters.

You can focus on architecting better systems, mentoring junior engineers, improving processes, and solving the interesting problems that drew you to this field in the first place.

The **dozens** of tools in my stack? Still there. But now I have one additional tool that helps me use all the others more effectively. And unlike the others, this one actually makes my job easier instead of more complicated.

So go ahead, pick a prompt, solve a problem, and reclaim a piece of your sanity. Your future self will thank you, probably from a nice vacation that you actually have time to take.

Start with prompt #1 and let me know in the comments which ones save you the most time. And if you create your own killer prompts, share them, we’re all in this together.

Found this helpful? Give it a clap and follow for more AI-powered DevOps tips that will change how you work. Because life’s too short for manual deployments and YAML indentation errors.

> 💌 **Join My Newsletter - One Lesson A Week**No jargon, No noise, Just practical tips to make DevOps less scary and more doable.
> 👉 **Join here*Lets Connect On Linkedin***

Thank You For Reading

📢 **DevOps Engineers:** Need troubleshooting commands at your fingertips? Check out my GitHub repo with comprehensive DevOps troubleshooting commands, scripts, and guides: DevOps-Troubleshooting-Toolkit. If you find it useful, a ⭐ would be appreciated!

Bonus: Want to see **real-world DevOps projects** that can help you land your dream job fast? Here’s my curated list !’ **DevOps Portfolio Projects**

💬 *Enjoyed this post or found it helpful?*
you can buy me a coffee here ☕

Other mentions by Author

• medium.com | Written by Zudonu Osomudeya