DevOps MCP Server

A Model Context Protocol (MCP) server that provides DevOps tools and resources for Kubernetes, Prometheus, Grafana, databases, Redis, and more.

Features

- **Kubernetes Integration**: List and manage Kubernetes resources across namespaces
- **Modular Architecture**: Easily extensible with new tools and services
- **Read-Only Operations**: Focus on retrieving information without making changes
- **MCP Protocol**: Compatible with Claude and other MCP-enabled AI assistants

Requirements

- Python 3.12 or higher
- Kubernetes configuration (kubeconfig file)
- Additional requirements based on enabled services (Prometheus, Grafana, etc.)

Quick Installation

Use the provided installation script to set up the project:

```
./install.sh
```

This script will:

1. Check if Python 3.12+ and uv are installed
2. Create and activate a virtual environment
3. Install dependencies
4. Make all scripts executable

PROF

Usage

Starting the Server

Start the server with the default configuration:

```
python main.py
```

Or specify a custom kubeconfig file:

```
python start_server.py /path/to/your/kubeconfig
```

Testing with MCP Inspector

You can test the server using the MCP Inspector tool:

```
npx @modelcontextprotocol/inspector python main.py
```

For services that require specific environment variables, you can pass them using the `-e` flag:

```
# For Kubernetes tools
npx @modelcontextprotocol/inspector -e
"KUBECONFIG=/path/to/your/kubeconfig" python main.py

# For AWS tools (using AWS profiles)
npx @modelcontextprotocol/inspector -e "AWS_PROFILE=your-profile-name" -
e "AWS_REGION=us-east-1" python main.py

# For multiple services
npx @modelcontextprotocol/inspector -e "KUBECONFIG=/path/to/kubeconfig"
-e "AWS_PROFILE=your-profile-name" python main.py
```

Project Structure

The project follows a modular architecture designed for extensibility and maintainability:

```
/devops-mcp-server/
├── main.py                 # Entry point for the MCP server
├── server.py               # Backward-compatible entry point
├── start_server.py         # Script to start the server with custom
config
├── setup_mcp_server.py     # Script to configure Claude to use this
server
├── install.sh              # Installation script
├── config/                 # Configuration files and settings
│   ├── __init__.py
│   └── settings.py         # Global settings and configuration
parameters
├── core/                   # Core server functionality
│   ├── __init__.py
│   └── exceptions.py       # Custom exceptions for error handling
├── services/               # Service implementations for different
tools
│   ├── __init__.py
│   ├── base.py             # Base service class and service registry
│   ├── kubernetes/         # Kubernetes service
│   │   ├── __init__.py
│   │   └── client.py       # Kubernetes client implementation
│   ├── prometheus/         # Prometheus service (future)
```

```
│   ├── grafana/             # Grafana service (future)
│   ├── database/            # Database services (future)
│   └── redis/               # Redis service (future)
├── tools/                   # MCP tools implementation
│   ├── __init__.py
│   └── kubernetes_tools.py  # Kubernetes tools for MCP
├── resources/               # MCP resources implementation
│   ├── __init__.py
│   └── kubernetes_resources.py # Kubernetes resources for MCP
└── utils/                   # Utility functions and helpers
    ├── __init__.py
    ├── config.py            # Configuration utilities
    ├── formatting.py        # Response formatting utilities
    ├── logging.py           # Logging utilities
    └── validation.py        # Input validation utilities
```

Directory and Module Descriptions

Root Directory Files

- **main.py**: The primary entry point for the MCP server. Initializes all components and starts the server.
- **server.py**: A backward-compatible entry point that delegates to main.py.
- **start_server.py**: A script to start the server with a custom kubeconfig file.
- **setup_mcp_server.py**: A script to configure Claude to use this MCP server.
- **install.sh**: Installation script for setting up the project environment.

Config Directory

The `config/` directory contains configuration files and settings for the server:

- **settings.py**: Defines global settings and configuration parameters for all services, including:
  - Server settings (name, version)
  - Kubernetes settings (kubeconfig path, timeout)
  - Prometheus settings (URL, timeout)
  - Grafana settings (URL, API key, timeout)
  - Database settings (connection parameters)
  - Redis settings (host, port, password)
  - Logging settings (level, format)

PROF

Core Directory

The `core/` directory contains core server functionality:

- **exceptions.py**: Defines custom exceptions for error handling, including:
  - DevOpsServerError: Base exception for all server errors
  - ServiceConnectionError: Exception for service connection failures
  - ServiceOperationError: Exception for service operation failures

- ResourceNotFoundError: Exception for resource not found errors
- InvalidArgumentError: Exception for invalid arguments

Services Directory

The `services/` directory contains service implementations for different tools:

- **base.py**: Defines the base service class and service registry:
  - BaseService: Abstract base class for all services
  - ServiceRegistry: Registry for managing service instances
- **kubernetes/**: Kubernetes service implementation:
  - **client.py**: Implements the KubernetesService class for interacting with the Kubernetes API
- **prometheus/**: Prometheus service implementation (future)
- **grafana/**: Grafana service implementation (future)
- **database/**: Database services implementation (future)
- **redis/**: Redis service implementation (future)

Tools Directory

The `tools/` directory contains MCP tools implementation:

- **kubernetes_tools.py**: Implements Kubernetes tools for the MCP server:
  - KubernetesTools: Class that registers Kubernetes tools with the MCP server
  - Tools include: list_k8s_resources, describe_k8s_resource, get_k8s_logs, list_k8s_namespaces

Resources Directory

PROF

The `resources/` directory contains MCP resources implementation:

- **kubernetes_resources.py**: Implements Kubernetes resources for the MCP server:
  - KubernetesResources: Class that registers Kubernetes resources with the MCP server
  - Resources include: k8s://namespaces, k8s://{namespace}/{resource_type}, k8s://{namespace}/{resource_type}/{name}

Utils Directory

The `utils/` directory contains utility functions and helpers:

- **config.py**: Configuration utilities for loading and managing configuration
- **formatting.py**: Response formatting utilities for MCP responses:
  - format_text_response: Format a text response

    - format_json_response: Format a JSON response
    - format_table_response: Format a table response
    - format_list_response: Format a list response
    - format_key_value_response: Format a key-value response
    - format_error_response: Format an error response

- **logging.py**: Logging utilities for consistent logging across the server:

    - setup_logger: Set up a logger with the specified name and level
    - log_exception: Log an exception with an optional message
    - log_request: Log an incoming request
    - log_response: Log an outgoing response

- **validation.py**: Input validation utilities for validating tool and resource parameters:

    - validate_required_args: Validate that required arguments are present
    - validate_string_arg: Validate a string argument
    - validate_int_arg: Validate an integer argument
    - validate_bool_arg: Validate a boolean argument
    - validate_enum_arg: Validate an enumeration argument
    - validate_list_arg: Validate a list argument
    - validate_dict_arg: Validate a dictionary argument

AWS Tools

The server provides a comprehensive set of AWS tools to interact with various AWS services:

AWS Service Categories

The MCP server includes tools for the following AWS services:

1. **Cost Explorer** - Analyze and monitor AWS costs
2. **S3 (Simple Storage Service)** - Manage S3 buckets and objects
3. **ECS (Elastic Container Service)** - Work with ECS clusters, services, and tasks
4. **ELB (Elastic Load Balancing)** - Manage load balancers and target groups
5. **VPC (Virtual Private Cloud)** - Work with VPCs, subnets, and security groups
6. **CloudFront** - Manage CloudFront distributions
7. **IAM (Identity and Access Management)** - Work with IAM roles and policies
8. **Lambda** - Manage Lambda functions
9. **RDS (Relational Database Service)** - Work with RDS instances
10. **ElastiCache** - Manage ElastiCache clusters
11. **SSM (Systems Manager)** - Work with SSM parameters and documents

AWS Cost Tools

The server provides several AWS Cost Explorer tools to analyze and monitor AWS costs:

Available Cost Tools

- **get_aws_cost_and_usage**: Get general cost and usage data
    - Parameters: start_date, end_date, granularity, metrics
    - Example: Get monthly cost data for the last 30 days
- **get_aws_cost_by_service**: Get costs grouped by AWS service
    - Parameters: start_date, end_date, granularity
    - Example: See which AWS services are costing the most
- **get_aws_cost_by_account**: Get costs grouped by AWS account
    - Parameters: start_date, end_date, granularity
    - Example: Compare costs across different AWS accounts
- **get_aws_cost_by_region**: Get costs grouped by AWS region
    - Parameters: start_date, end_date, granularity
    - Example: Analyze which regions are generating the most costs
- **get_aws_cost_forecast**: Get cost forecasts
    - Parameters: start_date, end_date, granularity, metric
    - Example: Predict future AWS costs
- **get_aws_cost_anomalies**: Get cost anomalies
    - Parameters: start_date, end_date, anomaly_monitor, anomaly_subscription
    - Example: Identify unusual spending patterns

Available Cost Resources

- **aws://cost/usage**: Get AWS cost and usage data
- **aws://cost/by-service**: Get AWS cost data grouped by service
- **aws://cost/by-account**: Get AWS cost data grouped by account
- **aws://cost/by-region**: Get AWS cost data grouped by region
- **aws://cost/forecast**: Get AWS cost forecast
- **aws://cost/anomalies**: Get AWS cost anomalies

PROF

AWS S3 Tools

Tools for working with Amazon S3 buckets and objects:

- **list_s3_buckets**: List all S3 buckets in your AWS account
- **get_s3_bucket**: Get details of a specific S3 bucket
- **list_s3_objects**: List objects in an S3 bucket with optional prefix filtering
- **get_s3_object**: Get metadata of a specific S3 object
- **get_s3_object_url**: Generate a presigned URL for an S3 object
- **get_s3_bucket_size**: Calculate the size of an S3 bucket

AWS ECS Tools

Tools for working with Amazon ECS clusters, services, and tasks:

- **list_ecs_clusters**: List all ECS clusters
- **get_ecs_cluster**: Get details of a specific ECS cluster
- **list_ecs_services**: List services in an ECS cluster
- **get_ecs_service**: Get details of a specific ECS service
- **list_ecs_tasks**: List tasks in an ECS cluster
- **get_ecs_task**: Get details of a specific ECS task

AWS ELB Tools

Tools for working with Elastic Load Balancers:

- **list_elb_load_balancers**: List all load balancers
- **get_elb_load_balancer**: Get details of a specific load balancer
- **list_elb_target_groups**: List target groups for a load balancer
- **get_elb_target_group**: Get details of a specific target group

AWS VPC Tools

Tools for working with Virtual Private Clouds:

- **list_vpc_vpcs**: List all VPCs
- **get_vpc_vpc**: Get details of a specific VPC
- **list_vpc_subnets**: List subnets in a VPC
- **get_vpc_subnet**: Get details of a specific subnet
- **list_vpc_security_groups**: List security groups in a VPC
- **get_vpc_security_group**: Get details of a specific security group

Required Environment Variables for AWS Tools

To use the AWS tools, you need to set the following environment variables:

PROF

- **AWS_PROFILE**: The AWS profile to use (configured in ~/.aws/credentials)
- **AWS_REGION**: The AWS region to use (e.g., us-east-1)

The server uses AWS profiles for authentication, which is more secure and convenient than using direct credentials. Make sure you have configured your AWS profiles using the AWS CLI (`aws configure`) or by manually editing the ~/.aws/credentials file.

Configuring in Roo Code Global MCP Settings

To add this MCP server to Roo Code's global MCP configuration, add the following to your `mcp_settings.json` file:

```
{
  "mcpServers": {
    "devops-server": {
      "command": "python",
```

```
      "args": ["/path/to/your/project/main.py"],
      "env": {
        "FASTMCP_LOG_LEVEL": "INFO",
        "AWS_PROFILE": "your-profile-name",
        "AWS_REGION": "us-east-1"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

You can customize the environment variables based on which tools you want to use:

```
{
  "mcpServers": {
    "devops-server": {
      "command": "python",
      "args": ["/path/to/your/project/main.py"],
      "env": {
        "FASTMCP_LOG_LEVEL": "INFO",

        "# For AWS tools": "",
        "AWS_PROFILE": "your-profile-name",
        "AWS_REGION": "us-east-1",

        "# For Kubernetes tools": "",
        "KUBECONFIG": "/path/to/your/kubeconfig",

        "# For Prometheus tools": "",
        "PROMETHEUS_URL": "http://localhost:9090",

        "# For Vault tools": "",
        "VAULT_URL": "http://localhost:8200",
        "VAULT_TOKEN": "your_vault_token",

        "# For Redis tools": "",
        "REDIS_HOST": "localhost",
        "REDIS_PORT": "6379",
        "REDIS_PASSWORD": "your_redis_password",

        "# For MongoDB tools": "",
        "MONGODB_URI": "mongodb://localhost:27017",

        "# For PostgreSQL tools": "",
        "POSTGRES_HOST": "localhost",
        "POSTGRES_PORT": "5432",
        "POSTGRES_USER": "postgres",
        "POSTGRES_PASSWORD": "your_postgres_password",

        "# For GitHub tools": "",
```

PROF

```
        "GITHUB_ACCESS_TOKEN": "your_github_token"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

Available Tools

Generic Kubernetes Resource Tool

Instead of creating separate tools for each Kubernetes resource type, we use a single generic resource tool that can handle any resource type dynamically. This approach is more flexible and maintainable.

The generic resource tool provides the following capabilities, designed to be similar to kubectl commands:

- `get_k8s_api_resources`: Get all available API resources in the cluster
- `list_k8s_resources`: List resources of a specific type (similar to `kubectl get <resource_type>`)
  - Shows only minimal information (names and basic metadata)
  - Automatically determines the correct API version for the resource type
  - Uses "default" namespace by default
  - Supports listing across all namespaces with the `all_namespaces` flag
  - Supports label selector filtering with the `selector` parameter
  - Supports field selector filtering with the `field_selector` parameter
  - Limits results to 100 by default (configurable)
- `get_k8s_resource`: Get detailed information about a specific resource (similar to `kubectl get <resource_type> <name> -o yaml`)
  - Automatically determines the correct API version for the resource type
  - Uses "default" namespace by default
  - Supports any resource type (pods, deployments, services, etc.)
  - Returns the complete YAML definition of the resource
- `get_k8s_resource_events`: Get events related to any Kubernetes resource
  - Automatically determines if the resource is namespaced
- `get_k8s_logs`: Get logs from a pod with advanced filtering options
  - Supports container selection
  - Supports tail lines limitation
  - Supports previous container logs
  - Supports time-based filtering
  - Supports grep-style pattern filtering
- `get_k8s_version`: Get the Kubernetes version information

Kubernetes Resources

- `k8s://namespaces`: List all Kubernetes namespaces

- `k8s://{namespace}/{resource_type}`: List resources of a specific type in a namespace
- `k8s://{namespace}/{resource_type}/{name}`: Get details of a specific resource

Extending the Server

The modular architecture makes it easy to add new services and tools:

1. Create a new service in the `services/` directory:
    - Implement a new service class that extends BaseService
    - Initialize the service client and implement required methods
2. Implement tools for the service in the `tools/` directory:
    - Create a new class that registers tools with the MCP server
    - Implement tool functions that use the service
3. Implement resources for the service in the `resources/` directory:
    - Create a new class that registers resources with the MCP server
    - Implement resource handlers that use the service
4. Register the tools and resources in `main.py`:
    - Initialize the service
    - Initialize the tools and resources with the service

Dynamic Resource Handling

The generic resource tool can handle any Kubernetes resource type dynamically, including:

- **Core Resources**: Pods, Services, ConfigMaps, Secrets, Namespaces, etc.
- **Workload Resources**: Deployments, StatefulSets, DaemonSets, Jobs, CronJobs, etc.
- **Networking Resources**: Ingresses, NetworkPolicies, etc.
- **Storage Resources**: PersistentVolumes, PersistentVolumeClaims, StorageClasses, etc.
- **RBAC Resources**: Roles, RoleBindings, ClusterRoles, ClusterRoleBindings, etc.
- **Custom Resources**: Any CRD-defined resources in the cluster

PROF

This approach eliminates the need to create separate tools for each resource type, making the codebase more maintainable and future-proof.

Installation via Package Registry

You can install and use the DevOps MCP Server directly from PyPI using either uv or uvx.

Installation with uv

To install the package using uv:

```
uv pip install msrashed100-devops-mcp
```

This will install the package and its dependencies in your current Python environment.

Installation with uvx

For a quick one-off execution without installing:

```
uvx msrashed100-devops-mcp
```

This will download and run the latest version of the DevOps MCP server.

Using as an MCP Server with Claude

After installation, you can configure Claude to use this as an MCP server:

```
{
  "mcpServers": {
    "devops-server": {
      "command": "msrashed100-devops-mcp",
      "env": {
        "FASTMCP_LOG_LEVEL": "INFO",
        "AWS_PROFILE": "your-profile-name",
        "AWS_REGION": "us-east-1",
        "KUBECONFIG": "/path/to/your/kubeconfig"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

PROF

License

MIT