RV College of Engineering®

Department of AIML

Machine Learning Operations

Implementation of ZenML

Faculty Mentor: Prof. S. Anupama Kumar

Team:	USN:
Pavithra C	1RV22AI038
Nischitha P	1RV22AI031
DV Sarayu Reddy	1RV22AI015
Akshita Chavan	1RV22AI005

Go, change the world®

Introduction to ZenML

ZenML is an extensible, open-source MLOps framework for creating portable, production-ready **MLOps pipelines**. It's built for data scientists, ML Engineers, and MLOps Developers to collaborate as they develop to production.

It enables MLOps infrastructure experts to define, deploy, and manage sophisticated production environments that are easy to share with colleagues.

Programming Languages that support the ZenML Framework: - Python

Key Features:

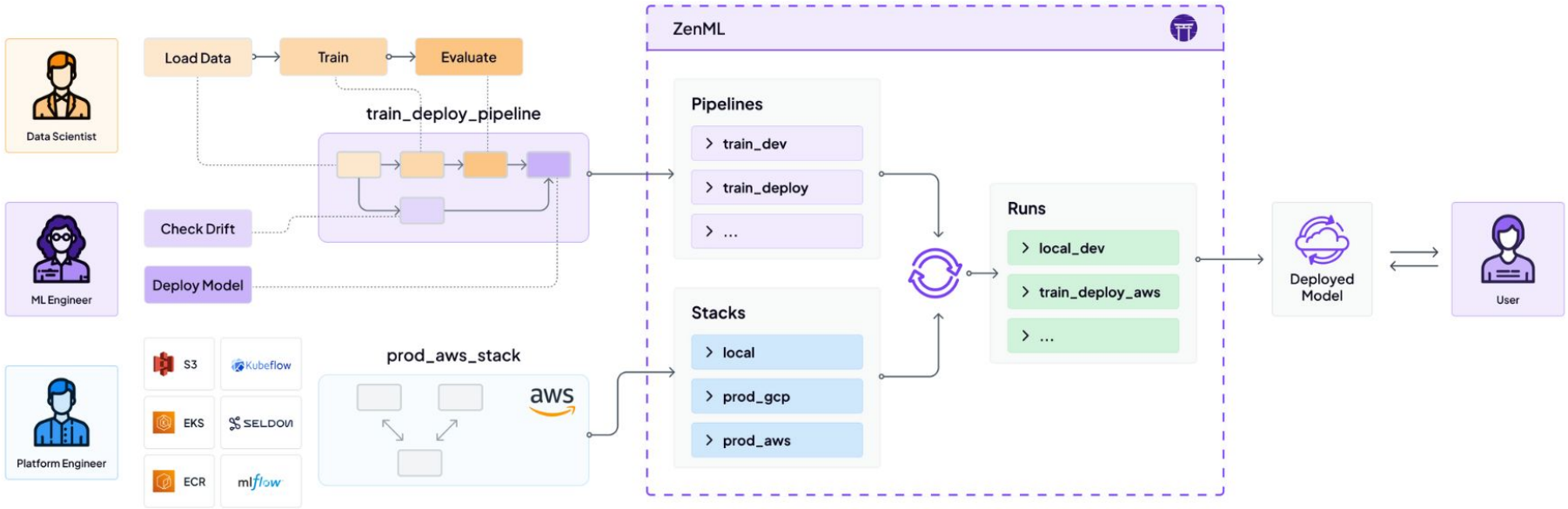
Pipeline Composition
Versioning & Reproducibility
Stack Management
Automation & Scheduling
Integration Ecosystem
Orchestrator Support:
Metadata Store
Custom Steps

ZenML can also manage **LLM (Large Language Model) pipelines**, enabling the orchestration, versioning, and tracking of LLM tasks such as data preprocessing, model training, fine-tuning, and deployment.

Key Components of ZenML

1. **Pipeline Composition**: Modular and reusable pipelines, enabling easy composition and sharing of ML workflows.
2. **Versioning & Reproducibility**: Strong focus on versioning artifacts, steps, and experiments for end-to-end reproducibility.
3. **Integration Ecosystem**: Seamless integration with a variety of third-party tools (e.g., TensorFlow, PyTorch, Kubeflow, MLflow).
4. **Metadata Store**: Centralized tracking of pipeline metadata, experiments, and artifacts for traceability and comparison.
5. **Custom Steps**: Users can define custom steps and workflows, making ZenML adaptable to any ML use case.
6. **Orchestrator Support**: Compatibility with multiple orchestrators like Apache Airflow, Kubeflow, and others for distributed pipeline execution.
7. **Stack Management**: Modular stack configuration to easily switch tools and infrastructure components in pipelines.
8. **Automation & Scheduling**: Built-in automation for pipeline execution, including scheduling features for recurring tasks.

Overview of Zen ML



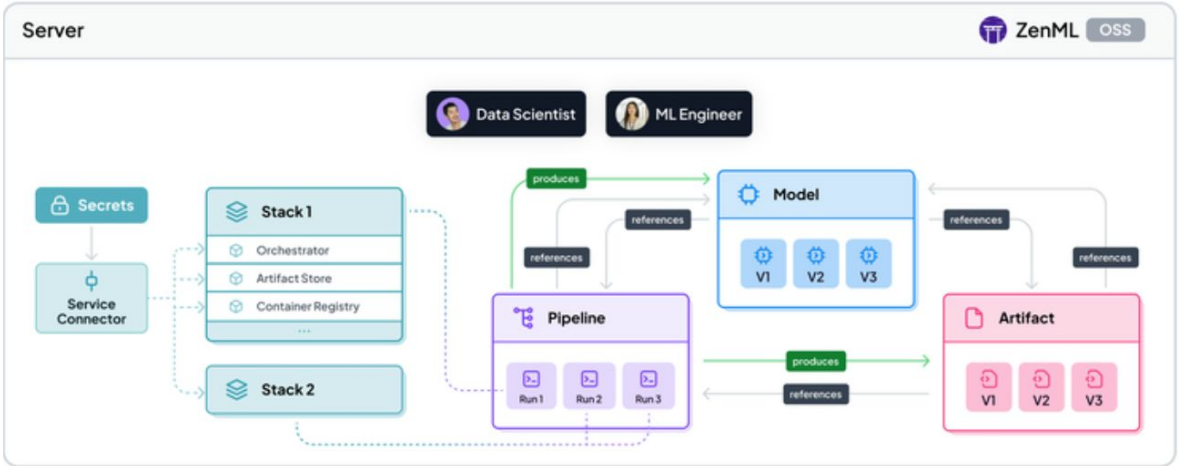
Core Concepts of ZenML

ZenML introduces various concepts for different aspects of an ML workflow and we can categorize these concepts under three different threads:

Development: As a developer how do i design my ML workflows?

Execution: While executing , how do my workflows utilise the large landscape of MLOPs infrastructure?

Management: How do i establish and maintain a production grade and efficient solution ?

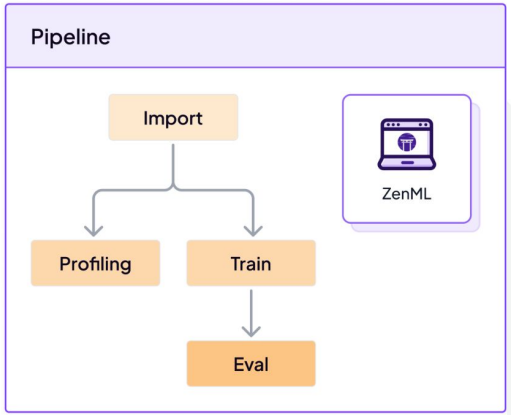


A diagram of core concepts of ZenML OSS

Core Concepts of ZenML

1.Development

Pipelines : At its core, ZenML follows a pipeline-based workflow for your projects. A pipeline consists of a series of steps, organized in any order that makes sense for your use case.



Pipelines and steps are defined in code using Python decorators or classes. This is where the core business logic and value of work lives.

Data flows through these steps in the form of inputs and outputs, which are tracked as artifacts. These artifacts, once created, are stored and versioned for easy traceability and reuse. ZenML also uses serializers to manage how data is converted between steps, ensuring smooth transitions and maintaining compatibility across the pipeline.

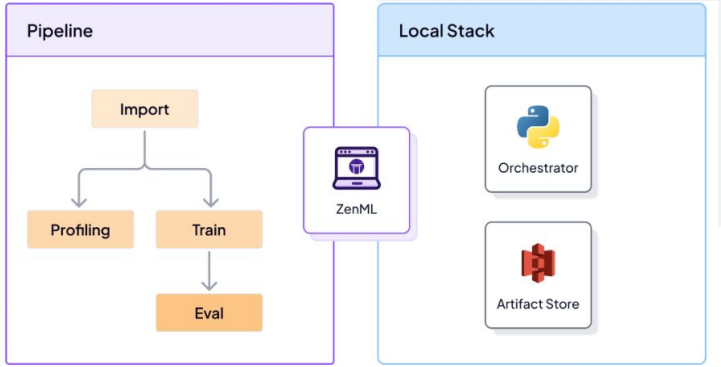
Core Concepts of ZenML

2. Execution

1. **Stacks & Components**: In ZenML, a Stack is a collection of components that configure various MLOps pipeline functions, like orchestration systems, artifact repositories, and model deployment platforms.

2. **Orchestrator**: The orchestrator manages pipeline execution by coordinating steps, handling dependencies, and determining the order and timing of step execution, especially useful during project exploration.

3. **Artifacts**: Artifacts are the inputs and outputs of pipeline steps, stored in an artifact store rather than passed in memory. Each step writes its outputs to storage, which are then loaded for subsequent steps.

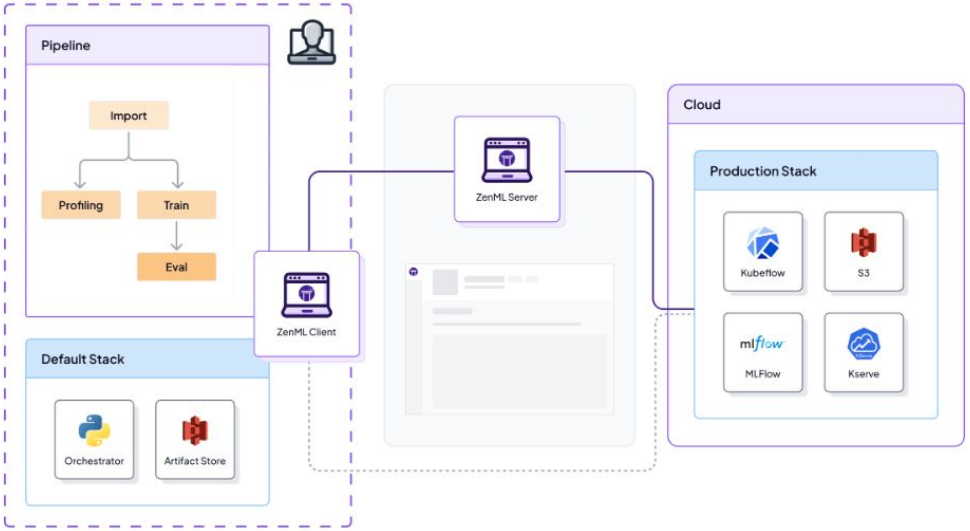


Core Concepts of ZenML

3. Management

In order to benefit from the aforementioned core concepts to their fullest extent, it is essential to deploy and manage a production-grade environment that interacts with your ZenML installation. ZenML Server To use stack components that are running remotely on a cloud infrastructure, you need to deploy a ZenML Server so it can communicate with these stack components and run your pipelines. The server is also responsible for managing ZenML business entities like pipelines, steps, models, etc.

Visualization of the relationship between code and infrastructure.



Core Concepts of ZenML

- **Artifacts**: Artifacts represent the data (inputs/outputs) passing through steps. They are stored in the artifact store and not passed between steps in memory. Once a step completes, artifacts are written to storage and loaded for the next step.
- **Models**: In ZenML, models include weights and associated data. They are first-class entities, centralized for easy management and access via the ZenML API, client, and Pro dashboard.
- **Materializers**: Materializers define how data is serialized/deserialized between steps. They handle loading input data and storing output, using the `BaseMaterializer` class, with built-in implementations for various data types.
- **Model Versions**: ZenML organizes models into versions, linking them to provide a unified view of all models and their iterations within a project.

Core Concepts of ZenML

The Pro version of ZenML

ZenML Pro adds a managed control plane with benefits like:

- **Managed production-grade ZenML deployment**
- **User management with teams**
- **Role-based access control and permissions**
- **Enhanced model and artifact control plane**
- **Triggers and run templates**
- **Early-access features**

ZenML Pro provides a control plane that allows you to deploy a managed ZenML instance and get access to exciting new features such as CI/CD, Model Control Plane, and RBAC.

Requirement Specification

Hardware Requirements:

- **CPU**: Modern multi-core processor (Intel i5 or equivalent, or higher recommended)
- **RAM**: At least 8 GB (16 GB recommended for large datasets and complex models)
- **Storage**: At least 50 GB of free disk space for storing datasets, artifacts, and model versions
- **GPU** (optional): A dedicated GPU (e.g., NVIDIA with CUDA support) is recommended for ML tasks that require heavy computation (e.g., training deep learning models)

Software Requirements:

- **Operating System**:
  - Linux (Ubuntu preferred)
  - macOS
  - Windows (via WSL or Docker for better compatibility)
- **Python**:
  - Python 3.9 or higher (latest stable release recommended)
- **Package Manager**:
  - `pip` (for installing Python packages)
  - Alternatively, `conda` can be used for managing environments
- **ZenML**:
  - ZenML installation via `pip` (`pip install zenml`)

Requirement Specification

- **Dependencies**:
  - **ML Frameworks**: TensorFlow, PyTorch, or other frameworks as needed for your ML models
  - **Docker** (optional but recommended for containerized environments and cloud integration)
  - **Git** (for version control of pipelines and artifacts)
- **Cloud Integration** (optional but beneficial):
  - Access to cloud platforms like AWS, Google Cloud, or Azure for large-scale training and model deployment
  - Relevant SDKs (e.g., `boto3` for AWS, `google-cloud` for GCP)
- **Orchestrators** (if using distributed/complex workflows):
  - Apache Airflow, Kubeflow, or other supported orchestrators (ZenML can integrate with these tools)

Setting Up ZenML

Installing ZenML: pip install zenml

Note: ZenML currently supports Python 3.9, 3.10, 3.11 and 3.12.

Install with the dashboard

1. ZenML comes bundled with a web dashboard that lives inside a sister repository. In order to get access to the dashboard **locally**, you need to launch the ZenML Server and Dashboard locally.
2. For this, you need to install the optional dependencies for the ZenML Server: **pip install "zenml[server]"**
3. At ZenML, We like to use virtualenvwrapper or pyenv-virtualenv to manage our Python virtual environments.

Setting Up ZenML

Installing onto MacOS with Apple Silicon (M1, M2)

A change in how forking works on Macs running on Apple Silicon means that you should set the following environment variable which will ensure that your connections to the server remain unbroken:
export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES
This environment variable is needed if you are working with a local server on your Mac, but if you're just using ZenML as a client / CLI and connecting to a deployed server then you don't need to set it.

Nightly builds

ZenML also publishes nightly builds under the `zenml-nightly` package name. These are built from the latest `develop` branch (to which work ready for release is published) and are not guaranteed to be stable. To install the nightly build, run:
Copy
pip install zenml-nightly

Setting Up ZenML

Deploying the server

Though ZenML can run entirely as a pip package on a local system, complete with the dashboard. You can do this easily:
pip install "zenml[server]"

zenml login --local # opens the dashboard locally

However, advanced ZenML features are dependent on a centrally-deployed ZenML server accessible to other MLOps stack components. You can read more about it here.

For the deployment of ZenML, you have the option to either self-host it or register for a free ZenML Pro account.

Setting Up ZenML

Verifying installations

Once the installation is completed, you can check whether the installation was successful either through Bash: **zenml version**
or through Python:
import zenml
print(zenml.__version__)

Running with Docker

`zenml` is also available as a Docker image hosted publicly on DockerHub. Use the following command to get started in a bash environment with `zenml` available:
docker run -it zenmldocker/zenml /bin/bash

If you would like to run the ZenML server with Docker:
docker run -it -d -p 8080:8080 zenmldocker/zenml-server

Flexible and Scalable MLOps Deployment

Self-hosted deployment: ZenML can be deployed on any cloud provider and provides many Terraform-based utility functions to deploy other MLOps tools or even entire MLOps stacks:

Connect cloud resources with a simple wizard zenml stack register --provider aws

Deploy entire MLOps stacks at once zenml stack deploy --provider gcp

Standardization: With ZenML, you can standardize MLOps infrastructure and tooling across your organization. Simply register your staging and production environments as ZenML stacks and invite your colleagues to run ML workflows on them.

Register MLOps tools and infrastructure zenml orchestrator register kfp_orchestrator -f kubeflow

Register your production environment zenml stack register production --orchestrator kubeflow ...

Flexible and Scalable MLOps Deployment

No Vendor Lock-In: Since infrastructure is decoupled from code, ZenML gives you the freedom to switch to a different tooling stack whenever it suits you. By avoiding vendor lock-in, you have the flexibility to transition between cloud providers or services, ensuring that you receive the best performance and pricing available in the market at any time.

zenml stack set gcp

python run.py # Run your ML workflows in GCP

zenml stack set aws

python run.py # Now your ML workflow runs in AWS

Benefits of Using ZenML

- **Experiment Tracking :** Log and visualize pipeline runs, metrics, artifacts, and parameters, enabling easy comparison of results across multiple experiments. ZenML ensures detailed tracking of each pipeline step for seamless evaluation and debugging.
- **Reproducibility:** Package pipelines with all dependencies, environments, and configurations to ensure reproducible machine learning workflows. This allows users to recreate experiments effortlessly, even months later, regardless of the environment.
- **Model Management :**Register, version, and deploy models seamlessly using ZenML's integrations with deployment platforms like MLflow and Seldon Core. Models are tracked as part of pipelines, ensuring a smooth transition from development to production.
- **Flexibility:** Supports multiple machine learning frameworks, including TensorFlow, PyTorch, and Scikit-learn, while allowing customization for unique workflows. Its tool-agnostic design makes it adaptable to various MLOps use cases.
- **Scalability:** Effortlessly scale pipelines and tracking infrastructure to handle increasing workloads, datasets, and team sizes. ZenML integrates with orchestration tools like Kubeflow and Airflow for large-scale, distributed ML workflows.
- **Collaboration:** Facilitates team collaboration by centralizing experiments, pipeline runs, and results. Teams can share pipelines, monitor progress, and improve transparency across the ML lifecycle, fostering better teamwork.

Drawbacks of ZenML

- ZenML is Python-centric, which may limit its adoption in organizations or teams that rely heavily on other programming languages like R or Julia.
- For large-scale, complex workflows with high concurrency, ZenML might not be as efficient as more mature tools like Apache Airflow or Kubeflow
- While ZenML is open-source, its unique abstractions and workflows could lead to a form of "tool lock-in." Switching to another MLOps tool might require significant rework.
- ZenML does not inherently provide advanced analytics or visualization for pipeline metrics, relying instead on integrations with third-party tools.

THANK YOU