

A) Write and share a small note about your choice of system to schedule periodic tasks (such as downloading a list of ISINs every 24 hours). Why did you choose it? Is it reliable enough; Or will it scale? If not, what are the problems with it? And, what else would you recommend to fix this problem at scale in production?

Ans: In our system, we have chosen to leverage Apache Airflow for scheduling periodic tasks, particularly for downloading a list of ISINs every 24 hours. Our decision is based on several key considerations that align with our system requirements.

Why Apache Airflow:

1. Workflow Orchestration:

- Apache Airflow provides robust workflow orchestration capabilities. This allows us to define, schedule, and monitor complex workflows, making it an ideal choice for handling tasks with dependencies, such as the periodic download of ISINs.

2. Scalability:

- Airflow is designed to scale horizontally, accommodating increased workloads by adding more workers. This ensures that our system can seamlessly handle growth in data and task complexity.

3. Task Dependency Management:

- Airflow's Directed Acyclic Graph (DAG) structure allows us to define and manage task dependencies effectively. This is crucial for ensuring that tasks are executed in the correct order, preventing issues in data consistency.

4. Extensibility:

- Airflow is extensible and supports a wide range of integrations. This flexibility allows us to adapt and extend functionality as our requirements evolve.

Considerations:

1. Learning Curve:

- While Airflow offers powerful features, there might be a learning curve for teams unfamiliar with its concepts. However, we believe that the long-term benefits outweigh the initial investment in learning.

2. Resource Management:

- Efficient resource management is essential for optimal Airflow performance. Monitoring and configuring resources based on task requirements are crucial considerations.

Recommendations for Scalability:

1. Optimized DAG Configuration:

- Regularly review and optimize the configuration of DAGs to ensure efficient task execution and resource utilization.

2. Scaling Worker Nodes:

- Proactively scale the number of worker nodes based on the increasing workload to maintain performance levels.

3. Monitoring and Alerts:

comprehensive monitoring and alerting systems to quickly identify and address any issues in task execution or system performance.

In conclusion, our choice of Apache Airflow is driven by its strengths in workflow orchestration, scalability, and task dependency management. We are confident that, with proper configuration and monitoring, it will effectively handle the periodic download of ISINs and scale as our system evolves.

B) In what circumstances would you use Flask instead of Django and vice versa?

Ans:

Everyone loves python and its amazing frameworks. Among the set of many Python frameworks, Flask and Django are the biggest talks of the town. Unsurprisingly, everybody is now curious about how Flask & Django are different and how to choose between them.

If you, too, are deciding on which one to use for your next web application development project, honestly, you will hardly go wrong with either framework. Both are extensible, mature web frameworks, which offer similar functionality in handling requests and serving up documents. However, both differ in their scope of responsibilities. While one is a full-stack framework, another is a power-packed micro-framework catering to the needs of modern web Dev solutions.

In the realm of web development, selecting the right framework is a critical decision that significantly influences the development process, scalability, and maintainability of your application. Flask and Django are two popular choices, each with its own strengths and use cases.

Flask: Lightweight Flexibility

1. Project Size and Complexity:

- **Scenario:** Flask excels in smaller to medium-sized projects, offering a lightweight and modular structure.
- **Benefits:** Flexibility to choose individual components allows for a tailored approach suitable for less complex applications.

2. Modularity and Customization:

- **Scenario:** Flask's "build it as you go" philosophy caters to developers who prefer a modular structure with the ability to choose specific components.
- **Benefits:** Customization is a key advantage, enabling developers to integrate only what is needed for a particular project.

3. Learning Curve:

- **Scenario:** Developers who want to quickly get started with web development without dealing with the complexities of a larger framework.
- **Benefits:** Flask has a smaller learning curve, making it accessible for developers new to web development.

Django: Opinionated for Rapid Development

1. Rapid Development and Convention Over Configuration:

- **Scenario:** Django excels in scenarios where rapid development is a priority, thanks to its "batteries-included" philosophy.
- **Benefits:** A comprehensive set of built-in features (ORM, authentication, admin panel) minimizes configuration hassles, allowing developers to focus on building features.

2. Built-in Admin Interface:

- **Scenario:** Applications with administrative or content management needs benefit from Django's powerful and customizable admin panel.
- **Benefits:** Saves development time by providing an out-of-the-box solution for managing application data.

3. Monolithic Applications:

- **Scenario:** Larger projects or monolithic applications with more complex requirements.
- **Benefits:** Django's structure and conventions provide a solid foundation for maintaining code consistency and collaboration among larger development teams.

General Considerations

1. Community and Ecosystem:

- **Scenario:** Consider the size and activity of the community and the availability of extensions.
- **Benefits:** Flask and Django both have active communities, but Django's ecosystem tends to be more extensive due to its monolithic nature.

2. Project Size and Complexity:

- **Scenario:** Tailor the choice based on the size and complexity of the project.
- **Benefits:** Flask for smaller projects; Django for larger, more complex applications.

3. Flexibility vs. Opinionated Structure:

- **Scenario:** Developers' preference for flexibility or a more opinionated structure.
- **Benefits:** Flask offers more flexibility, whereas Django provides a structured approach for easier collaboration.

In conclusion, the choice between Flask and Django depends on factors such as project size, development preferences, and the balance between flexibility and convention. Flask is a versatile and customizable choice, while Django excels in providing a robust, all-inclusive framework for rapid development.