**Problem Set 3**

**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)**

For scheduling periodic tasks like downloading a list of ISINs every 24 hours in a Django application, one common choice is to use Celery along with Celery Beat. Celery is a distributed task queue that allows you to run background tasks asynchronously, while Celery Beat is a scheduler that enables you to schedule periodic tasks at specified intervals.

**Why did you choose it? Is it reliable enough; Or will it scale?**

Using Celery with Celery Beat offers several benefits for managing periodic tasks like downloading a list of ISINs every 24 hours. Celery is known for its reliability and scalability, making it suitable for production environments where tasks need to be executed efficiently and at scale without impacting application performance. Its integration with Django allows for seamless setup and management of background tasks within Django applications. Celery Beat provides a convenient way to schedule tasks with a simple syntax, adding flexibility to task execution by allowing prioritization, retry policies, and progress monitoring. Overall, Celery with Celery Beat offers a reliable, scalable, and flexible solution for managing periodic tasks in Django applications.

**What are the problems with it?**

While Celery with Celery Beat is a robust solution for scheduling periodic tasks in Django applications, it does have some limitations and challenges, especially when it comes to scaling in production.

**Resource Management**: As the workload increases, managing Celery workers and ensuring optimal resource allocation becomes more challenging. Configuring the right number of workers, adjusting concurrency settings, and monitoring resource usage are crucial for maintaining performance and scalability.

**What else would you recommend to fix this problem at scale in production?**

To address resource management challenges with Celery at scale in production, several strategies can be employed:

1. **Dynamic Scaling**: Implement dynamic scaling mechanisms that automatically adjust the number of Celery workers based on workload and resource usage metrics. This can involve using tools like Kubernetes or AWS Auto Scaling to scale worker instances up or down as needed.

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

Use Flask When:

1. **Small to Medium-sized Projects**: For simple or small-scale projects with fewer built-in features and a focus on customization, Flask's simplicity and minimalism can be advantageous.
2. **Learning or Prototyping**: Flask's simplicity and minimalistic design make it suitable for beginners or for rapid prototyping where quick iteration and experimentation are priorities.

Use Django When:

1. **Large-scale Projects**: For large-scale projects requiring robustness, scalability, and maintainability, Django's comprehensive ecosystem and conventions can streamline development and ensure consistency.
2. **Security and Authentication**: Django's built-in security features, including robust authentication mechanisms and protection against common web vulnerabilities, make it suitable for applications requiring high levels of security.