Q1. 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?

My choice for scheduling periodic tasks would be Celery, a distributed task queue framework for Python. Celery allows you to run tasks asynchronously in the background, and schedule them to run at specific times or intervals.

The reasons for choosing Celery are as follows:
- Reliability: Celery is a mature and widely used framework with a large community of contributors, and has been tested and proven to be reliable in production environments.
- Scalability: Celery is designed to scale horizontally across multiple worker nodes, so it can handle large workloads and can be used in a distributed environment.
- Flexibility: Celery supports a variety of brokers and result backends, such as RabbitMQ, Redis, and others. This allows you to choose the one that best fits your specific use case.
- Integration: Celery integrates well with many popular web frameworks and libraries, such as Django and Flask.

However, there are some potential issues with Celery in production environments, such as scalability limitations when using a single broker. To address these issues, it may be necessary to use a more powerful broker or backend, such as RabbitMQ or Redis, and carefully monitor and optimize your Celery configuration.

In summary, Celery is reliable.

Q2. In what circumstances would you use Flask instead of Django and vice versa?

Both Flask and Django are popular web frameworks for building web applications in Python, but they have different strengths and weaknesses. Choosing one over the other depends on the specific requirements and constraints of your project. Here are some circumstances in which you might choose one over the other:

Use Flask when:
- You need a lightweight framework that allows for more flexibility in terms of how you structure your application.
- You have a small to medium-sized project with a simpler structure that doesn't require a lot of built-in features.
- You need to build a RESTful API or microservices.
- You want to use a modular architecture where you can choose the components you need and build a custom stack of tools and libraries.
- You have experience with Python and want to build a web application using the Python ecosystem.

Use Django when:
- You need a full-stack framework with a lot of built-in features for authentication, routing, templating, ORM, and other common web development tasks.
- You have a larger project with a complex structure that requires a lot of built-in functionality and features.
- You need to build a content management system or an e-commerce platform.
- You want to use a batteries-included framework with a lot of third-party libraries and packages built on top of it.
- You want to take advantage of Django's built-in admin interface for managing data and content.

In summary, Flask is a good choice for smaller projects that require more flexibility and a simpler structure, while Django is a better choice for larger, more complex projects that require a lot of built-in functionality and features. Ultimately, the choice between the two frameworks depends on your specific requirements and constraints.
le and scalable choice for scheduling periodic tasks, but it's important to carefully consider your specific use case and requirements, and optimize your configuration for performance and scalability.