1.

Rest API	MCP
User have to call each endpoint one by one: For example: POST /api/generate_images GET /api/images	Instead of services, we can have tools connected to our services and the model/agent will call themselves For example, if it needs to generate images it will run generate_images
Can be used on AI-based systems (Even though it's not optimal). It can also work without LLM.	Can only be used if LLM is involved

2. I assume this question means that "How REST API & MCP can improve AI use cases"

REST API	MCP
It could handle cases involving calling models, etc. It could serve as a basic framework. However, when it comes to advanced AI-based systems like RAG, REST couldn't provide an optimal solution. For example, we have to create every endpoint for every services. We also have to ensure that every LLM has connection to each endpoint itself.	On the other hand, MCP provide a more efficient framework. It acts as an orchestrator that could route queries from user to the relevant services or AI agents. This approach is more favorable to be used as a base framework for AI-based systems.

3. Depends on what kind of task our AI Agents capable for, but there are several approach that is quite common to use to avoid hallucinations:
    a. If the task require our model to interact with huge amount of data, it is better to use RAG as our data source or fetch other API to provide data. It gave us ground truth for the model.
    b. Uses structured output like JSON, to enforce the agent to only communicate within limited parameters.
    c. If the parameter is available, set the temperature in the model into 0.1

4. Docker (or containerization) is important to ensure that our AI system works. It list down all of the necessary versions of the library that we need and it's easier to "transport" elsewhere. Instead of installing each version one by one, ensuring that all of the version matches with each other, we can just build the docker image and use them.

    It also allow us to interact with services that have different requirement. Let's say we have a LLaMa model that needs to run in the latest Python version, while our vector DB can only run in Python 3.5. Containerization allow to do that communicating through the port.

5. Assume the data is in a form of text and in English, assume that we already have a base model that is capable to be run on our GPU/system.

    a. **Preprocessing:** We have to ensure that the data is properly pre-processed first. Since the quality of data hugely impact the quality of the fine-tuned model results. We have to ensure that it has the necessary attributes and covers all of the possible use cases that we need for our specific task.

    We also have to consider how many raw data that we have in our hand? If its possible, consider getting a lot of data or even creating new ones.

    b. **Splitting the data** into training, validation & test. It is also best to decide what kind of evaluation metrics that is important for this task & target threshold.
    c. Since we already decided on the base model, we can continue to focus on the **fine-tuning approach.**
        a. If you want the model to learn new domain knowledge -> LoRA, QLoRA
        b. If you want the model to act a certain way or give personality -> Prompt Tuning
        c. If you want the model to have interchangeable new skills (you can turn it off & on during certain times) -> Adapter
    d. Fine-tune the model using the preferred approach and our dataset. Also apply the evaluation metrics during the fine-tuning process.
    e. If a certain threshold is reached. Stopped the fine-tuning process & test them on test data.
    f. The fine-tuned model is done.