How to Schedule Scripts

The Artifacts Needed:

- **Scheduled Job**: A Python function that you wish to run at specific intervals.
- **Hooks**: A mechanism to tell Frappe when to run your scheduled job.

Crafting the Scheduled Job:

1. **Create a Python Function**: Define a Python function that performs the desired task in your custom Frappe app.

   Example:

   ```
   # my_app/my_app/tasks.py
   import frappe

   def awaken_the_dark_magician():
       frappe.publish_realtime('msgprint', message='The Dark Magician has been awakened!', user=frappe.ses
   sion.user)
   ```

   Here, `awaken_the_dark_magician` is a simple function that sends a message to the user. Your function can contain any logic that you wish to execute.

Enchanting the Hooks:

1. **Declare the Scheduled Job in Hooks**: In your app, navigate to the `hooks.py` file and declare your scheduled job.

   Example:

   ```
   # my_app/hooks.py
   scheduled_jobs = [
       {
           "doctype": "Dark Magician",
           "cron": "0/5 * * * *",
           "function": "my_app.my_app.tasks.awaken_the_dark_magician",
           "args": (),
           "kwargs": {},
           "disabled": False
       }
   ]
   ```

   - `"cron": "0/5 * * * *"`: This cron syntax tells Frappe to run the job every 5 minutes. Adjust the cron expression according to your desired schedule.
   - `"function": "my_app.my_app.tasks.awaken_the_dark_magician"`: This points to the function you wish to run.
   - `"args"` and `"kwargs"`: If your function requires arguments, provide them here.
   - `"disabled": False`: Ensure this is set to `False` to enable the job.

   After changing any scheduled events in hooks.py, you need to run bench migrate for changes to take effect.

Additional Scrolls:

▼ **Config/Settings**

In Frappe, the scheduler can be enabled or disabled at two levels: for a specific site or for all sites (globally). Here's how you can manage it:

1. Enabling/Disabling Scheduler for a Specific Site:

If you want to enable or disable the scheduler for a specific site, you should use the `site_config.json` file which is located in the respective site's directory.

To Enable:

```
{
    "pause_scheduler": false
}
```

To Disable:

```
{
    "pause_scheduler": true
}
```

2. Enabling/Disabling Scheduler Globally:

If you want to control the scheduler for all sites, you should use the `common_site_config.json` file, which is located in the `frappe-bench/config` directory.

To Enable:

```
{
    "pause_scheduler": false
}
```

To Disable:

```
{
    "pause_scheduler": true
}
```

Notes:

- **Priority:** If the scheduler is disabled at the site level (in `site_config.json`), it will be disabled for that site even if it is enabled globally in `common_site_config.json`.
- **Use Case:**
    - Use `site_config.json` if you want to control the scheduler for individual sites independently.
    - Use `common_site_config.json` if you want to apply a setting to all sites, for instance, if you're performing server maintenance and want to pause the scheduler for all sites.

Commands:

You can also use bench commands to enable or disable the scheduler:

For a Specific Site:

```
# To Disable
bench --site [site_name] disable-scheduler

# To Enable
bench --site [site_name] enable-scheduler
```

Globally:

```
# To Disable
bench disable-scheduler
```

```
# To Enable
bench enable-scheduler
```

Conclusion:

Choose the configuration level that best suits your use case. If you're managing multiple sites and want to control the scheduler for each one individually, use `site_config.json`. If you want to enforce a global policy for all sites, use `common_site_config.json`. Always remember to check both configurations to ensure the scheduler is behaving as expected.

- **Bench Commands**: Use the following bench commands to manage and troubleshoot scheduled jobs:
  - `bench enable-scheduler`: To enable the scheduler.
  - `bench disable-scheduler`: To disable the scheduler.
  - `bench doctor`: To check the status of the scheduler and background workers.

For a Specific Site:

```
# To Disable
bench --site [site_name] disable-scheduler

# To Enable
bench --site [site_name] enable-scheduler
```

Globally:

```
# To Disable
bench disable-scheduler

# To Enable
bench enable-scheduler
```

Final Words of Wisdom:

- **Testing**: Test your scheduled job in a safe environment before deploying it to production.
- **Logs**: Keep an eye on the Scheduler Log (via Desk > Developer > Scheduler Log) to ensure that your tasks are running as expected and to catch any errors.

Example: Scheduling integrate function

To run a function at specific times using Frappe's scheduler, you'll need to define a scheduler event in your Frappe app. Here's a step-by-step guide:

Step 1: Ensure Scheduler is Enabled

Make sure the scheduler is enabled in your Frappe setup. Ensure that `"pause_scheduler": false` or omit the key in your `site_config.json` or `common_site_config.json`.

Step 2: Define Scheduler Event

In your Frappe app, you need to define the scheduler events in the `hooks.py` file. If you want to run `scheduled_import` function from `integrate_from_cloud.py` every day at 6am and 6pm, you should define a cron job in your `hooks.py` file.

Here's an example of how you might set this up in your `hooks.py`:

```
scheduler_events = {
    "cron": {
        # syntax: ("cron_expression", "path.to.function")
        ("0 6,18 * * *", "your_app_name.path.to.integrate_from_cloud.scheduled_import"),
    }
}
```

Replace `your_app_name` with the actual name of your app, and `path.to.integrate_from_cloud` with the actual path to your `integrate_from_cloud.py` file.

Explanation:

- `"0 6,18 * * *"`: This cron expression tells the scheduler to run at minute 0 of hour 6 and 18 (6am and 6pm) every day of the month, every month, and every day of the week.
- `"your_app_name.path.to.integrate_from_cloud.scheduled_import"`: This should be the path to your `scheduled_import` function.

Step 3: Migrate and Restart

After adding the scheduler event, you should migrate your site and restart the bench to apply the changes:

```
bench --site your_site_name migrate
bench restart
```

Replace `your_site_name` with the name of your site.

Notes:

- Ensure that your function doesn't have any errors, as it might fail silently in the scheduler. Always test your function by calling it manually to ensure it works as expected.
- Check the scheduler logs in Frappe from time to time to ensure that your function is running as expected. You can find this under "System" -> "Scheduler Log" in your Frappe instance.
- If you're using a production setup, ensure that the scheduler is running in your supervisor configuration. If you're using developer mode, simply use `bench start` to start the scheduler.

This setup should allow your `scheduled_import` function to run at 6am and 6pm every day.

Step 4: Testing the Spell

Before entrusting your function to the scheduler, invoke it manually to ensure it works as expected:

```
bench execute pentaho_processes.scripts.integrated_transformation_log_data.fetch_and_insert
```

If the function runs without issues, the scheduler should have no trouble invoking it.

Step 5: Observing the Magic

Once deployed, keep an eye on the **Scheduler Log** in your Frappe Desk (`Developer > Scheduler Log`). This log will record the outcomes of scheduled tasks, allowing you to identify if your function encounters any issues when run by the scheduler.

Step 6: Troubleshooting

Should you encounter issues:

- Ensure the path to your function in `hooks.py` is accurate.
- Check the **Error Log** (`Developer > Error Log`) in the Frappe Desk for any issues during execution.
- Ensure there are no syntax errors in your `hooks.py`.

With these steps, your `fetch_and_insert` function should be invoked by the scheduler at 6am and 6pm daily, ensuring that your data is consistently fetched and inserted without manual intervention.

▼ Separate Scheduled Jobs Based on Site

In Frappe, the `hooks.py` file is common for all sites that use a particular app. If you specify two scheduled jobs in `hooks.py`, both jobs will be scheduled for all sites that have the app installed.

However, if you want to conditionally run scheduled jobs based on the site, you can achieve this by adding a check within the scheduled function itself. Here's how you can do it:

1. **Define your scheduled jobs in `hooks.py`:**

```
scheduler_events = {
    "cron": {
        "*/5 * * * *": [
            "myapp.tasks.scheduled_job_1",
            "myapp.tasks.scheduled_job_2"
        ]
    }
}
```

2. **Add a conditional check within each scheduled function:**

```
# in myapp/tasks.py

import frappe

def scheduled_job_1():
    # Only run for Site A
    if frappe.local.site == 'site_a.local':
        # Your logic for scheduled_job_1
        pass

def scheduled_job_2():
    # Only run for Site B
    if frappe.local.site == 'site_b.local':
        # Your logic for scheduled_job_2
        pass
```

By using `frappe.local.site`, you can determine the current site context. The scheduled job will then only execute its logic if the current site matches the specified site.

This approach allows you to use a common `hooks.py` for all sites but conditionally execute logic based on the site context.

Depending on your needs and preferences, there are several alternative approaches you can consider:

1. **Separate Apps for Each Site**:
   - If the logic for the two scheduled jobs is significantly different, you could consider creating two separate Frappe apps, each with its own `hooks.py` and scheduled job. Install the appropriate app on each site.
2. **Site Config Check**:
   - You can add custom configurations in the `site_config.json` for each site. For example:

   ```
   {
     "run_scheduled_job_1": true,
     "run_scheduled_job_2": false
   }
   ```

   - In your scheduled job function, check the site config before executing:

   ```
   def scheduled_job_1():
       if frappe.conf.run_scheduled_job_1:
           # Your logic here
   ```

3. **Database Configuration**:
   - Store the configuration in a custom DocType, say "Site Scheduler Configuration". Here, you can have fields

  indicating which jobs to run.
  - In your scheduled job function, fetch the configuration from the database and execute accordingly.
4. **Environment Variables**:
   - Use environment variables to control which jobs run on which sites. This is especially useful if you're using containerization (like Docker).
   - Set an environment variable for each site and check this variable in your scheduled job functions.
5. **Custom Script**:
   - Instead of using the built-in scheduler, you can write a custom script that uses the Frappe API to determine which site it's running on and then calls the appropriate scheduled job. This script can be run via a system cron job or another task scheduler.
6. **Use Tags or Naming Conventions**:
   - If the site names follow a certain pattern or have specific tags, you can use string matching in your scheduled job functions to determine which logic to run.

Each of these approaches has its own advantages and trade-offs. The best approach will depend on your specific requirements, the complexity of your setup, and your personal or organizational preferences.